AFIT/GSO/ENG/92D-04

Retinal Modeling: Segmenting Motion from Spatio-Temporal Inputs using Neural Networks.

THESIS

David E. Swanson
Captain

AFIT/GSO/ENG/92D-04

Approved for public release; distribution unlimited

AFIT/GSO/ENG/92D-04

Retinal Modeling: Segmenting Motion from Spatio-Temporal Inputs using Neural Networks.

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Space Operations)

David E. Swanson, B.S.E.E.

Captain

December, 1992

Preface

Congratulations on selecting this thesis to review. The work contained herein is the culmination of twelve months of research. It deals with using a a massively parallel system modeled after the retina. The model, as it is referred to, identifies motion in its field-of-view and is built on equations originally documented by Stephen Grossberg and subsequently modified by H. Öğmen and S. Gagné in their 1990 article on the fly's visual system. This thesis is written with the idea that it should provide enough information to fully support any work based on its results. Therefore, this document is rather lengthy. Much of its bulk is contained in the appendices. Appendix A is provided for those not formally schooled in the biology of neurons and the retina. Consider it a brief, topical discussion of important facts about the retina. Of course a full treatment of the subject would include a review of all my references and more. The remaining appendices, minus the last, contain computer code. The last is a reference documenting the tools I found useful when working with images. The quality of this thesis is directly affected by those who influenced me over the last year.

Major Steve Rogers, my primary advisor was the most helpful. He was able to direct me to an area of research that suited my desires and talents. Throughout the year, he nudged me along the thesis route and provided vital guidance when I stalled. Much of the success of this work belongs to him. Major John Borsi and Captain Dennis Ruck were of great help in defining my research objectives and using the computers at AFIT. I thank them for their willingness to read and critique my thesis. Over the last few quarters I turned to a variety of professionals for assistance when I could not see the forest for the trees.

I thank Major Rice for his literary talents which I used very early in my literature review stage. Dr. Quinn and Dr. Oxley gave of their time to show me how to solve time varying differential equations, help that went a long way in creating my simulation software. Major Howell was more than willing to clue me in on the various simulation packages available for Unix, DOS,

and Machintosh operating environments. Dr. Ridgely helped me understand the implications of the models I analyzed and extended my knowledge of Fourier, Laplace, and Z transformations. If this were a speech I would give all these people a great round of applause for their help. Before I get off the subject, I must mention my family.

I thank God for the support my family is capable of extending to me every day of my life. My parents, Donald and Rae are both directly responsible for my success in this work. They provided me with the basic skills that lead to my selection for and completion of a masters degree. I thank my sister Lori Rae and brothers Donald and Robert for their support of my military career which takes me to distant locations but can never break the bonds that tie us. To my brother Robert, I must make special mention for his present of a hard drive, without which, I could never have created and maintained the volumes of data generated for this thesis. To my children, who do not, as of yet, understand why their father spends long hours away from them. I hope that the time I stole from them both leads, eventually, to a better tomorrow. Finally, I thank daily the love, patience, and support provided by my best friend, my confidant, my wife, Cynthia.

David E. Swanson

Table of Contents

List of Figures

List of Tables

Abstract

We applied two first-order, linear, time-varying, differential equations to the task of segmenting motion from sequences of images. The equations are modified Grossberg formulas for long-term and short-term memory models characterizing the neurotransmitter and cell-activity levels of a synapse and neuron. We described how a two layered, sensory, neural network can be built using the equations to simulate the amacrine neurons of the retina. The model is defined using adaptive input nodes (adaptive model) and is compared to a similar model without these nodes (O&G model). By replicating the basic amacrine neuron model to form both one- and two-dimensional arrays, we created a novel method for processing images over time and space. To simulate the veto effect observed in shunt inhibitory synaptic junctions, we applied a nonrecurrent, asynchronous, inhibitory region in the receptive field of our amacrine neural model. We show how this effects the performance of the model in one dimension. In two dimensions we investigate the models' response to synthesized imagery (pristine) and to real, forward looking infrared radar (FLIR) images. The output of our models are further processed through two types of moving-average filters – causal and noncausal.

Forward

> Space is clearly our most challenging frontier. Enroute to Mars, we will explore the Moon, advance Earth sciences, and develop new, innovative technologies. We will tap lunar, Martian, and solar energy sources as we explore the heights of human talent and ability. Along the way, America's drive, initiative, ingenuity, and technology - all those things that have made our nation the most successful society on Earth - will propel us toward a future of peace, strength, and prosperity. The challenge is before us.
>
> Thomas P. Stafford
> Chairmen, Synthesis Group
> Report on America's Space Exploration Initiative

Appropriately titled, "America at the Threshold", the Synthesis Group's report on America's Space Exploration Initiative (SEI) studied ways of extending Man's dominance of the solar system to the Moon and Mars. The report identified key technologies required to support future manned mission. Two such technologies are telerobotics and telepresence [Synthesis, 1991:77]. In a more detailed look at an unmanned mission to Mars, the Graduate of Space Operations Class of 1992 made an indepth study of a scientific, precursor mission to Mars [Amrine, 1992]. In their report, they emphasized the need for systems capable of carrying out experiments semi-autonomously. Systems that can visually interrogate their environment and make decisions solely based on visual information are of great importance to the SEI.

Although we currently have the capability to build expert systems that can handle the job on Mars, they are often bulky and inefficient. In light of current launch costs ($4,000 per pound) [Wilsey, 1991:13], every effort must be made to reduce the size and weight of our space-faring, scientific equipment. Therefore, we can not accept bulky vision systems and their support computers as solutions to unmanned, deep-space missions. One might ask,"Why not just wait for smaller and faster digital computers?" For SEI this may be an option, however a more earthly need demands that we search for other more efficient ways to gather visual data.

The world may have become a safer place with the fall of Russian communism but the ballistic missile threat is still very real due to the proliferation of nuclear technologies to aggressive third-

world nations (Iraq is one such example). To counter the threat of missile attack from a variety of sources, the Air Force is exploring technologies vital to missile defense. One area of concern is missile seekers – the part of an interceptor missile or surveillance system that detects a missile in the boost or post-boost phase. A missile seeker may be employed as an integral part of the interceptor or as a remote, weapons-control system placed in a strategic location – space. Because of the short flight times involved in missile deployment, the detector portion of a seeker must provide realtime information to the interceptor's guidance and control subsystems. Despite the great advantages offered by digital computers, they lack the ability to process data quickly enough for missile defense. Even the parallel architectures used in digital computers today lack the speed required by missile seekers. Solutions to the problem of real-time image processing will come from emerging technologies based on neural networks.

Neural-network research attempts to mimic the biological precedence set by the human brain and nervous system. Like the brain and nervous system, neural networks can be classified into sensory and computational or decision-making elements. The study of object identification using perceptrons in layered networks is a member of the latter category. Again, like the brain, the decision networks never work with *raw* sensory data. The data must be preprocessed to filter noise and unwanted information while providing a set of relevant input features from the visual, audio, and tactile senses. The advantages of neural networks come from their massively parallel nature and our ability to faithfully create them in electronic hardware. Returning to the problem of missile seekers, both sensory (vision) and decision networks promise great improvements over conventional technologies. Our thesis work deals with the sensory type neural network and vision research.

The work described in this thesis focuses on one small aspect of vision research – motion segmentation. However, its importance as an integral part of technological advancement can not be overstated. We take great pride in presenting this material and in the way of explaining our

enthusiasm for this work, we restate what Sir James Jeans said some 60 years earlier about our work in the eyes of posterity.

> We are living at the very beginning of time. We have come into being in the fresh glory of dawn, and a day of almost unthinkable length stretches before us with unimaginable opportunities for accomplishment. Our descendants of far-off ages, looking down this long vista of time from the other end will see our present age as the misty morning of human history. Our contemporaries of today will appear as dim, **heroic figures who fought their way through the jungles of ignorance, error, and superstition to discover truth.**
>
> Sir James Jeans, Astronomer, 1930

Retinal Modeling: Segmenting Motion from Spatio-Temporal Inputs using Neural Networks.

I. Introduction

Man's visual organs make use of optics to focus what we see onto a paper thin lining called the retina. The retina is a complex tangle of interconnected neurons, which transform visual stimuli into nerve impulses representing tonic (static) and phasic (changing) imagery. Man's study of the retina is aimed at gaining insight into the design of artificial visual systems that can process spatial and temporal information without delay (real-time). The term *early vision* was coined to describe work that focuses on modeling the retina. The term means that before the brain receives any visual data, the gathered images are heavily preprocessed to adjust intensity and contrast, and to provide motion information.

Motion detection, the subject of this research, can be performed in a variety of ways, most of which involve intensive use of digital computers. The simplest method of separating moving objects from the stationary background (motion segmentation) is to perform a frame to frame subtraction of pixel values. The problem with this simple technique is that a change in light intensity over an object's surface will also look like motion. More sophisticated techniques are used to segment motion in a scene but rely on image processing software that correlates filters over each image of each frame - this is a slow process at best. An alternative to digital image processing is to use an analog computer operating in a massively parallel configuration. In other words, using artificial neural networks based on the retina.

In this thesis we apply a neural model to the task of enhancing moving or transient image elements while suppressing stationary elements. Our aim is to process two-dimensional, temporal images in a way that passes only the transient portions of the image sequence.

1.1 Background and Terminology

Our model depends on the work of many talented scientists, only a few of which are cited here. Names like Frank Werblin [Werblin, 1973], Richard Masland [Masland, 1986], Tomaso Poggio [Poggio, 1987], and Christof Koch [Poggio, 1987] are all associated with experimental research into neural and retinal biology. It is from their work that we gain an understanding of image processing in the retina and appreciate the analytic expressions developed by Grossberg [Grossberg, 1987], Öğmen [Öğmen, 1990], and Gagné [Öğmen, 1990]. Ultimately, the transformation from empirical data to system fabrication comes to a close when theoretical models are turned into hardware by people like Carver Mead [Mead, 988], Misha Mahowald [Mahowald, 1991], and M. A. Massie [Massie, 1992]. This process, a kind of technology transfer pipeline, repeats and with each new cycle, a little more of biology is turned into an artificial neurosystem.

The most recent development from the pipeline is a microchip that emulates the retina's top three layers. The Silicon Retina as it is called, was produced at Caltech by Mead and his associate Mahowald [Mead, 1988]. Similar to the retina, the Silicon Retina is a type of focal plane processor (FPP) which places the computational power for image processing at the focal plane of an optics system. Mead's chip represents a major breakthrough in early vision research. The Silicon Retina exhibits the ability to adapt to stationary images and respond over a wide range of intensities as does the retina. Mead showed that FPP is possible in a limited sense under laboratory conditions. Following Mead's lead, Amber Engineering has taken the Silicon Retina design and is fabricating a more robust FPP called the Neuromorphic Infrared Focal Plane processor (NIFPp). The purpose of the NIFPp is to support missile target and tracking systems for the Air Force. Additionally, the NIFPp can act as an input preprocessor when testing models of the final three layers of the retina. We can then return to the technology pipeline and work on the next generation Silicon Retina – a chip that emulates the retina from input to output.

Frank Werblin, working with the mudpuppy salamander [Werblin, 1973:71] (Necturus Maculosus[1]), recorded the effects of a moving image on the lower layers of the retina. He found that the receptive field (annular regions of inputs to a single cell that provide opposing effects on the response of the cell) not only supports stationary image processing in the upper layers of the retina, but is a key component to motion processing. In fact, nearly half of ganglion cells respond to image motion [Levine, 1985:102]. At the time of his paper, the receptive field was considered to be a single structure that permeated the entire retinal meat. Some 13 years later, Richard Masland would dissect the amacrine layer to define a different reality.

Richard Masland identified amacrine cells by the type of neurotransmitter it produced. Using a special staining technique, Masland was able to categorize individual cells and show significant differences in their dendritic tree structure [Masland, 1986:108]. For the first time there was evidence that the amacrine cell, a primary element of the visual pathway was more a group of cells than a single type. Masland identified four amacrine cell types and suggests that as many as 26 others may exist [Masland, 1986:106]. The amacrine cell group forms multiple pathways within the plexus[2] between amacrine and ganglion soma[3] which are thought to sharpen the transient response of ganglia to changes in light [Masland, 1986:104]. It is in this plexus that the receptive field is found to have directional selectivity (response to motion in a preferred direction and no response in the opposite or null direction) to motion.

Directional selectivity is the result of a difference in signal propagation times and silent or shunt inhibitory synapses [Poggio, 1987:47]. Tomaso Poggio and Christof Koch built computer simulations of signals propagating through the retina. Using these simulations, they showed that if an excitatory signal moved more quickly through the retina than the inhibitory signal, it would induce a more intense response in the target ganglion. If the inhibitory signal was given a head

[1] The mudpuppy was chosen for its relatively large retinal cells. In 1973, micropipettes were rather large and would damage many other animal's cells.

[2] Plexus "1: a network of anastomosing or interlacing blood vessels or nerves. 2: an interwoven combination of parts or elements in a structure or system." - Webster's Ninth Collegiate Dictionary.

[3] Soma "1: the body of an organism. ... 3: CELL BODY" - Webster's Ninth Collegiate Dictionary.

start, it would arrive in the ganglion along with the quicker excitatory signal and they would cancel. Therefore, image motion in a direction that creates inhibitory signals ahead of excitatory ones would not elicit a response while motion in the opposite direction would. The concept of directional selectivity is extremely important and will be addressed in more detail in Chapter II. The next stage of the our technology pipeline is to convert empirical data and theories into mathematical models.

In 1987, Grossberg published a technical description of many neural-network models both linear and nonlinear [Grossberg, 1987]. His article stands as the basis for many theoretical and mathematical treatments of neural networks. Later in this report we will look at two differential equations that describe long-term memory and short-term memory in neurons. We do this because they are used by H. Öğmen and S. Gagné to describe a motion sensitive model of the fly's eye.

Öğmen and Gagné reported on two separate but similar subjects in their 1990 article on motion perception [Öğmen, 1990]. In part one of their article, they describe a system that mimics the fly's visual system with an emphasis on response in one of two directions. This directionally sensitive model makes use of Grossberg's short-term and long-term memory equations with shunt inhibition. In part two, they examine two methods of delaying the inhibitory signal to the cell – recurrent and nonrecurrent. Using the nonrecurrent delay mechanism, Öğmen and Gagné illustrate the advantages of the asynchronous (delayed) model over the synchronous (not delayed) model [Öğmen, 1990:497]. These models form the basis for our research and will be presented in far more detail in Chapters II and III. However, before we get into any detail concerning the Grossberg or Öğmen and Gagné models, we will define our research, its scope, objectives, and limitations.

1.2 Research Definition

1.2.1 Research Questions. Because we base our research on the model defined by Öğmen and Gagné, we develop a software simulation designed by the author to investigate motion segmentation in sequences of images. Specific questions to be answered are;

1. How does the model respond to pristine image sequences?
2. What are the major features of the model's response to these inputs?
3. Can the model operate on images acquired by Forward Looking Infrared Radar image sequences?
4. What is the effect of moving the image's field-of-view with a moving object?
5. Can the output be further processed to better segment motion?
6. How do causal and noncausal filtering effect the model output?

1.2.2 Problem Statement. We will investigate the Öğmen and Gagné model response and compare it to the response of a derivative of their model called the adaptive model (which we will be defined in Chapter II). The images used in this project are of two kinds. One set represents a pristine image (no noise) and is produced by the author specifically for this research. The second image sequence is an excerpt from a FLIR video tape of a tank moving around in a field. We will examine the response of our models to each of the segments. The software used to simulate the neural network was developed by the author and is given in Appendix C.

1.2.3 Objectives. In order to answer the research questions posed above, we defined the following project objectives.

1. Examine the model response in regards to leading and trailing edges, time variant stimuli, and velocity and direction of motion.
2. Compare the response of the two-dimensional models with adaptive and nonadaptive inputs.
3. Compare the response of our models to both the pristine and FLIR image sequences.
4. Post process the response data of the models to enhance the transient portions of the images.

1.2.4 Scope. This research uses equations defined by Öğmen and Gagné to create one- and two-dimensional models through which sequences of images are processed. We look at methods of post processing the model response to obtain cleaner output. The emphasis here is on implementation of the equations and modification of the model such that it enhances motion and suppresses stationary objects.

1.2.5 Limitations.

1. Comparison of this model to other motion segmentation techniques will not be addressed.
2. We do not intend to add the post processing steps as integral parts of the software simulation.
3. Only a single set of FLIR images will be processed.
4. We will not fully explore how the model parameters effect the response of the models.

1.3 Thesis Overview

This thesis is organized into five chapters. Having read the first and prefatory material, we begin with the second. Chapter II is meant to be a focused literature review. Focused in that we discuss only those aspects of the retina that directly bear on our work with modeling the motion sensitive portions of the retina. In Chapter II we define three important concepts used in the retina to preprocess motion information - receptive fields, shunt inhibition, and directional selectivity. We also discuss the mathematical basis for our model as defined by Grossberg and briefly present the model for motion perception in the fly's eye as described by H. Öğmen and S. Gagné.

In Chapter III we examine the model's salient features. In the first sections of the chapter we discuss the theoretical underpinnings of each level of the model and explain how the model is put together. In the second half of Chapter III we look at some results obtained from early model simulations. Here we explain some post processing techniques that can be applied to the response data for enhancing the images. Chapter IV contains the results of model simulations using both the pristine and FLIR imagery. In the last chapter we provide an overall summary of the thesis results and findings.

It is our intention to provide more than enough information in this thesis so that future students might reproduce and extend our work. Therefore, we include information that does not directly impact our research but provides supplemental background in the appendices. Appendix A is designed for the uninitiated reader and provides a much broader look at the neuron, the retina, and retinal emulators than will be found in Chapter II. Appendix C contains the software code used to form our model. Since we spent some time finding out how to capture and work with images on our computers, we felt it important to document the procedures which are not readily available. This is provided in Appendix D. This last appendix also contains various points of contact which may be useful to future students.

II. Literature Review

2.1 Scope of the Review and Method of Treatment

This thesis concentrates on developing a model for motion in the latter layers of the retina. A full understanding of the retina is not critical to understanding the equations and computer model described in later chapters. Therefore, this chapter is designed to give the reader pieces of information that combine to support our model development. For those readers who wish to see more background material, we provided an extended literature review as Appendix A.

2.1.1 Organization. We begin with a brief review of retinal architecture in which the concepts of receptive field, shunt inhibition, and directional selectivity are defined. Naturally we follow the concept definitions with the analytic theory describing them. First we look at two models defined by Grossberg [Grossberg, 1987] as the short-term and long-term memory models. Then we examine the altered forms of these models defined by Öğmen and Gagné for the fly's visual system. These take the form of the neurotransmitter and cell-activity model. In the last portion of this chapter, we see how the cell-activity model can be generalized to two dimensions while adding inhibitory delays. When we finish, we will be ready to apply our version of these models to image sequences.

2.2 The Retina.

Figure 2.1 illustrates the three-dimensional nature of the retina. Consisting of five specialized neurons, the retina samples visual imagery at more than 126 million spacial locations by as many photoreceptors (rods and cones)[Lavine, 1985:75]. The photoreceptors transform photon (light) energy into electrical potentials (signals) and pass these to the network of neurons that follow. From a systems point of view, the retina is the largest vector processor known to man. A full explanation of each cell type and its operation in the retina is provided in Appendix A. What is

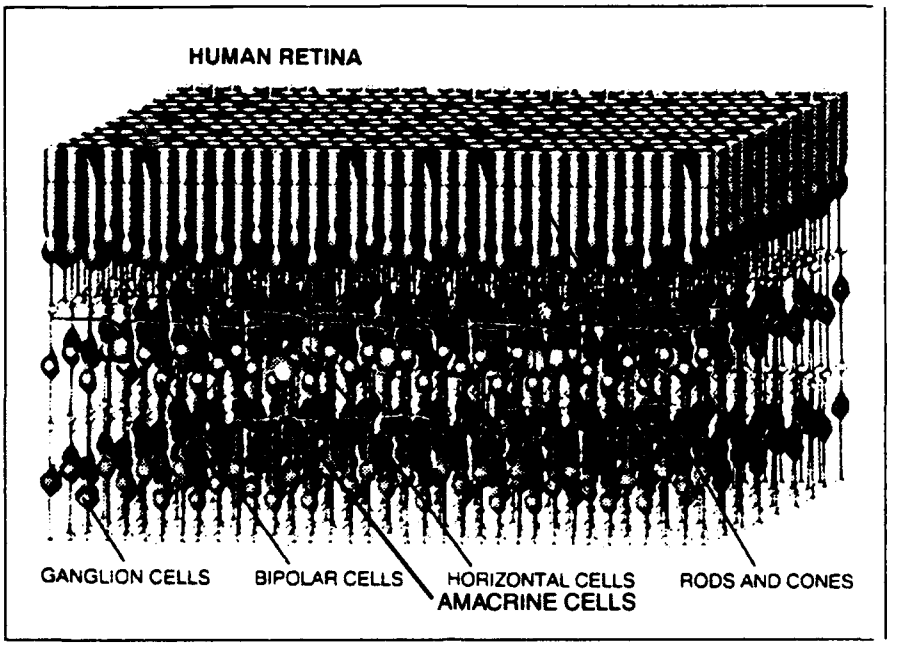

Figure 2.1. The Retinal Architecture [Mahowald, 1991:78]

important to recognize here is that motion processing is performed in the last layers of the retina: the amacrine and ganglion layers.

The amacrine cells provide a variety of paths oriented horizontally to the flow of image signals and represent a family of cell types that respond to changes in signal strength at the output of the bipolar cells. The ganglion cells (ganglia) receive signals from both the bipolar and the amacrine neurons. The signals are represented as slow potentials or time varying voltages. The ganglia transform the slow potentials transmitted by these neurons into action potentials (a series of voltage spikes) that are transmitted along the optic nerve. Ganglia action potentials represent both tonic and phasic information. To produce the latter, the ganglia depend on the amacrine cells to form receptive fields (subsection 2.2.1) which use shunt inhibition (subsection 2.2.2) and produce directionally selective outputs (subsection 2.2.3).

2.2.1 Receptive Field. The retina depends heavily on a structure called the receptive field. A neuron who's output depends on the activity of neighboring cells through lateral interconnections

forms a receptive field. In more concrete terms, the receptive field used in the retina has a central, circular region called the *center* and an outer annular region called the *surround.* There are two basic types of receptive fields, *on-center, off-surround* and *off-center, on-surround.* The two types correspond to a field which enhances lighted centers against dark surrounds (on-center) or enhances dark centers against lighted surrounds (off-center). Figure 2.2 illustrates an on-center receptive field. The left half of the figure shows an illuminated excitatory center and the cell's time response to this input. The right half of the figure shows an illuminated inhibitory surround and the cell's response over time. Receptive fields are used in the top three retinal layers to adjust visual images

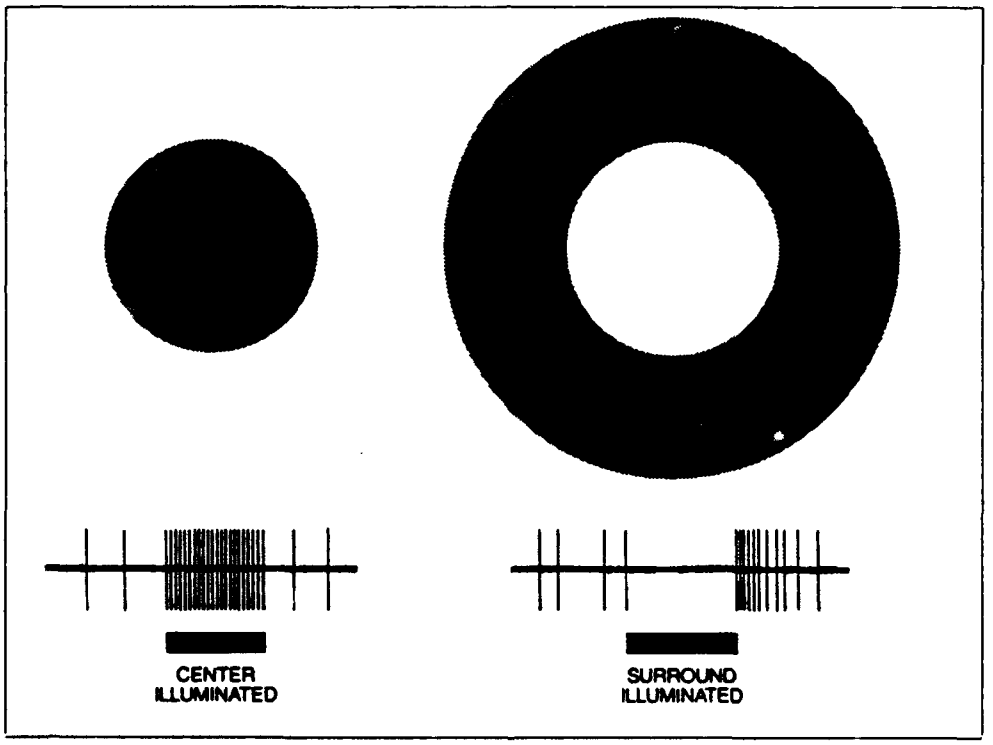

Figure 2.2. The On-Center Receptive Field [Masland, 1986:105]. This figure illustrates the firing pattern of a ganglion cell in the presence of a purely-excitatory signal from the fields center (left), and a purely-inhibitory signal from the surround (right). Below each example is a plot over time of the ganglion response. The black bar represents the duration of the stimulus. In the excitatory case, the cell responds for the duration of the stimulus. However, in the inhibitory case, the ganglion responds to the removal of the stimulus. Because the neuron is capable of adapting to stimuli, a removal of the inhibitory signal is identical to an introduction of an equally intense excitatory signal. The response dies off as the cell adapts once again to the input.

containing light from a wide band of light frequencies (10 logarithmic decades wide) to fit into the narrow frequency range of the bipolar cell (1.5 logarithmic decades wide). The lower two layers of the retina use receptive fields to detect motion.

2.2.2 Shunt Inhibition. So far we talked about a single type of inhibitory response. Neurons can use a different inhibitory scheme to mediate the response of the post-synaptic neuron. A scheme involving the shunt-inhibitory synapse. In the most common case, the inhibitory signal causes a hyperpolarization of the post-synaptic neuron while the excitatory signal causes a polarization of the neuron. If both signals are present, the response of the post-synaptic neuron is dependent on the relative strengths of the two opposing signals. Unlike the 'normal' inhibitory synapse, the shunt-inhibitory synapse balances the ionic concentration gradient across the post-synaptic membrane against the potential gradient over the membrane (Figure 2.3). When an inhibitory signal reaches the synapse and causes a channel to open in the membrane, the opposing gradients resist motion of ions into the post-synaptic neuron. Therefore, the neuron is unaffected. However, if a synapse, somewhere near the inhibitory synapse receives an excitatory signal, ions will enter the post-synaptic cell and change the gradients for a short time. If the shunt-inhibitory synapse receives a signal during this time period, it's channels will open and ions will now be allowed to flow into the post-synaptic neuron due to the unbalanced gradients [Poggio, 1987:49].

2.2.3 Directional Selectivity. Directional selectivity is an important aspect of the retina's motion processing. Research has shown that certain ganglia respond to motion only in a preferred direction and fail to respond to motion in the opposite or null direction [Masland, 1986:105]. Neurons that exhibit this behavior are called directionally selective. The mechanism used to support directional selectivity is the receptive field that uses a shunt inhibitory synapse [Poggio, 1987:49]. To illustrate how directional selectivity works, Figure 2.4 shows [Masland, 1986:105] a receptive field with a preferred direction to the right. In the figure, the left two diagrams show motion in the preferred direction to which the ganglion has a vigorous response (the cell response is shown

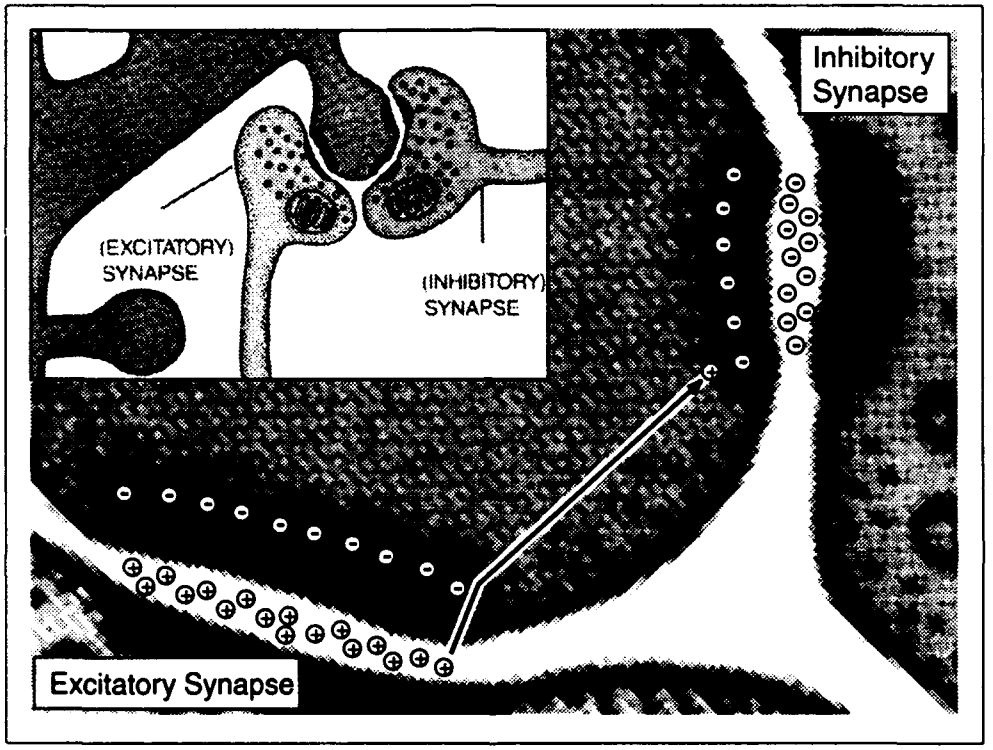

Figure 2.3. Shunt Inhibition. This figure shows two synaptic junctions located close together. The junction on the left is a typical synapse with its ion pumps ready for an excitatory signal. The synapse on the right is a shunt-inhibitory synapse where the ionic concentration exterior to the cell is much greater than that interior to the cell. However, both interior and exterior ions have negative charge. When a channel on the right is opened, the concentration gradient is balanced against the potential gradient. When the synapse on the left causes positive ions to enter the interior of the cell, the resulting change in potential at the right synapse allows negative ions to enter the cell and counter act the effect of the excitatory signal (Insert taken from [Goldman-Rakic, 1992:115]).

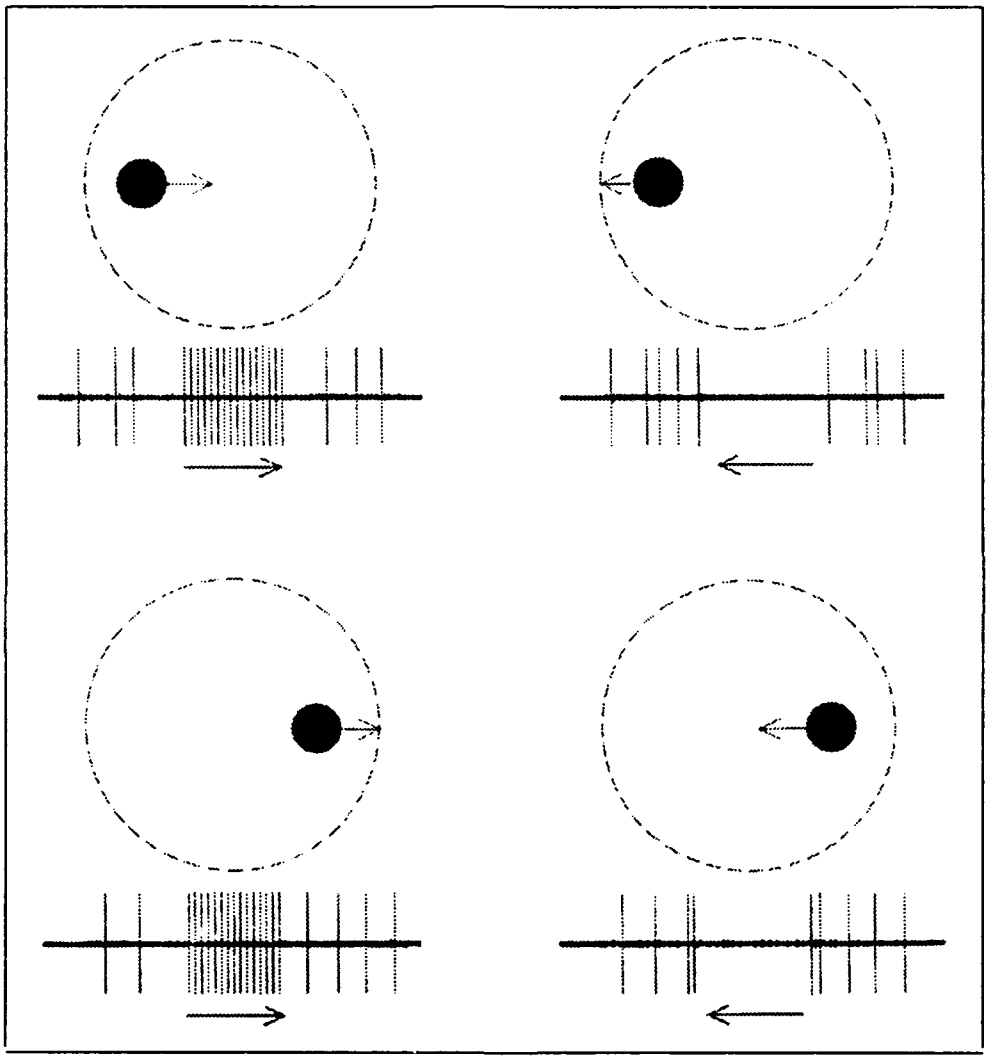

Figure 2.4. A Directionally Selective Receptive Field [Masland, 1986:105]. This figure illustrates how a directionally selective receptive field will respond to motion in a preferred direction (to the right). In the left two diagrams, the cell responds to motion to the right (preferred direction) regardless of its location. In the right two diagrams, the cell does not respond to motion to the left (the null direction) regardless of its location.

as action potentials along the time axis below each diagram). In the right two diagrams, motion is in the null direction to which the ganglion has no response. Random firing occurs in all neurons and is shown in all the time lines outside the movement period.

The receptive field is a four dimensional structure – two spacial dimensions, one hierarchical dimension (down through the retinal sheet), and time. Figure 2.4 shows the receptive field in the two spacial dimensions oriented tangential to the retinal surface. To understand how directional selectivity works, we need to look at how the signals propagate through the retinal sheet. Figures 2.5 and 2.6 [Poggio, 1987:50] are a series of snapshots showing the cross-sectional view of the retina's neural hierarchy. Figure 2.5 illustrates the propagation of signals due to an image moving in the

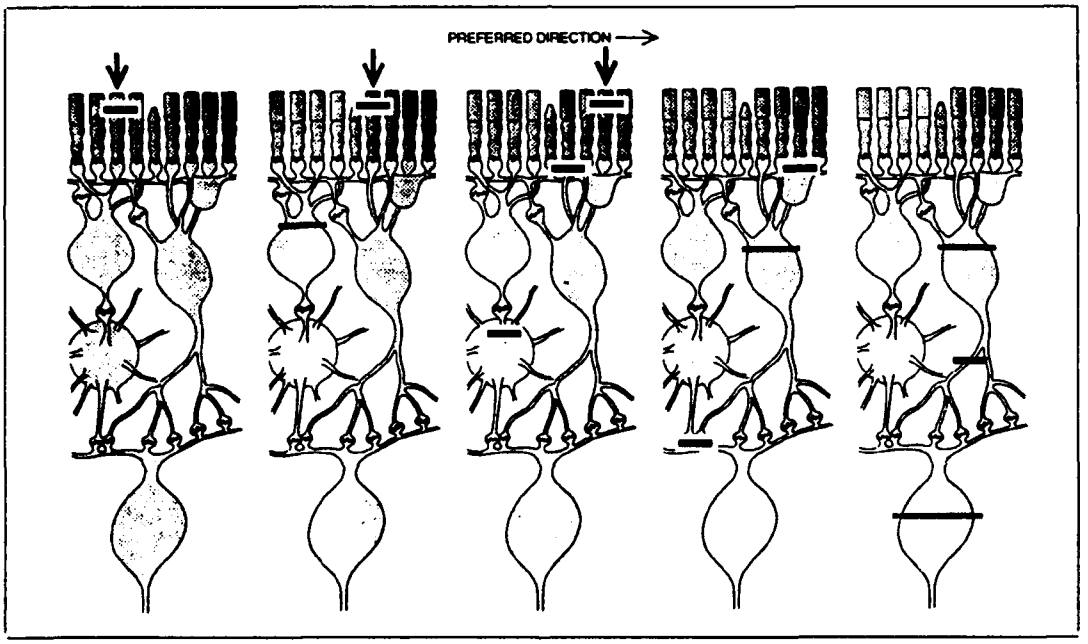

Figure 2.5. A Directionally Selective Receptive Field (cross section) [Poggio, 1987:50]. Here we have five snapshots of the retinal neuron hierarchy. When motion is in the preferred direction, the excitatory signal is created before the inhibitory signal. Since inhibitory signals propagate more slowly than excitatory ones, the two do not reach the ganglion at the same time. The ganglion, which uses shunt inhibition responds only to the excitatory signal.

preferred direction (This sequence of snapshots is arranged from left to right according to increasing time steps). In the preferred direction, the excitatory signal is created first and the inhibitory signal

follows. They both propagate through the retina but arrive at the ganglion at different times. Since ganglia use shunt inhibition, the cell responds only to the excitatory signal. In Figure 2.6, the image moves in the null direction. Here, the inhibitory signal reaches the synapse (snapshot on the left)

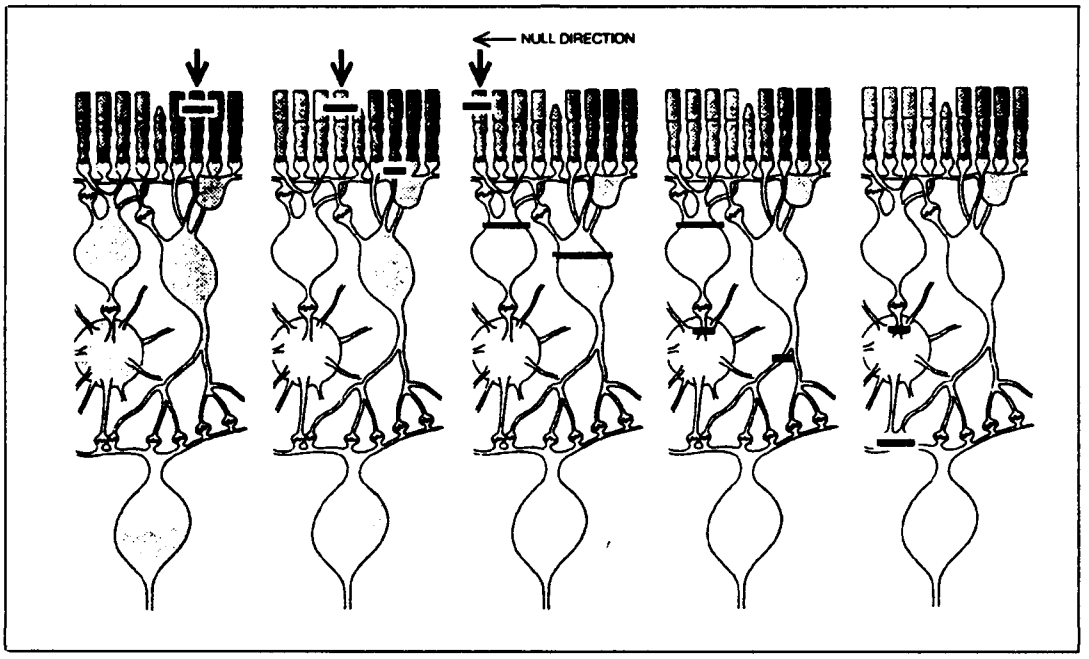

Figure 2.6. A Directionally Selective Receptive Field (cross section) [Poggio, 1987:50]. Here too, we have five snapshots of the retinal neuron hierarchy. When motion is in the null direction, the inhibitory signal is created before the excitatory signal. Since the excitatory signal moves through the retina more quickly than the inhibitory signal, the two meet at the ganglion simultaneously. The ganglion cell is effected by both signals and its response is a function of the relative strengths of the two signals.

at the same time the excitatory signal does. Therefore, the inhibitory signal cancels the excitatory signal at the ganglion cell. Shunt inhibition is easily modeled using differential equations.

2.3 Continuous-Nonlinear Network Mathematical Theory.

In 1987, Stephen Grossberg wrote a summary of the "unifying principles, mechanisms, and mathematical methods that arise in [neural network] theory" [Grossberg, 1987:1]. Included in his paper was a summary of the shunting, short-term memory (STM) trace equation and the passive-decay, long-term memory (LTM) trace equation. The concept of a trace stems from adaptive

behavior of the neural mechanism and has been used in many applications since the 1960's. In this report, the word *trace* means the output or curren: value of a state or cell.

2.3.1 The Short-Term Trace Equations. The adaptive STM equation can be expressed as [Grossberg, 1987:6];

$$\frac{dx_i}{dt} = -Ax_i + \left[\sum_{j=1}^{n} f_j(x_j)B_{ji}z_{ji}\right]^{(+)} - \left[\sum_{j=1}^{n} g_j(x_j)C_{ji}z_{ji}\right]^{(-)} + I_i \tag{2.1}$$

In Equation 2.1 the STM trace, x_i, is defined by a nonlinear, first-order, differential equation. The first term in the equation, $-Ax_i$, represents the spontaneous or passive decay of the trace with time. The second term, $\left[\sum_{j=1}^{n} f_j(x_j)B_{ji}z_{ji}\right]^{(+)}$, is the excitatory feedback and the third term, $\left[\sum_{j=1}^{n} g_j(x_j)C_{ji}z_{ji}\right]^{(-)}$, the inhibitory feedback.[1] The last term, I_i, is the input to the trace. Subscripts used in this equation indicate the principle cell, i, and the surrounding cells, j. Both the excitatory and inhibitory regions can incorporate many inputs, z_{ji}. Although the equation is summed in one dimension from 1 to n, an extra summation could be added to indicate an operation over some two-dimensional spacial region. The feedback terms involve state-dependent, nonlinear signals, $f_j(x_j)$ and $g_j(x_j)$, which are often implemented as sigmoidal functions [Grossberg, 1987:7] of the form;

$$S(x_j) = \frac{1}{1 + exp(\frac{x_j - x_i}{b})} \tag{2.2}$$

Figure 2.7 contains a plot of the sigmoid function. The values B_{ji} and C_{ji} are connection weights[2] often modeled as gaussian curves representing the excitatory and inhibitory regions of the receptive fields. Figure 2.7 shows a common form for these gaussians. The variable z_{ji} is the long-term memory trace of neighboring cells and will be discussed in the next section. This equation can be simplified by allowing the feedback terms and connection weights to be lumped together into

[1] The superscript (+) and (−) are used to indicate that these are either excitatory and inhibitory terms.

[2] The connection weights are also called path strengths implying that they are dependent on the two interacting cells.

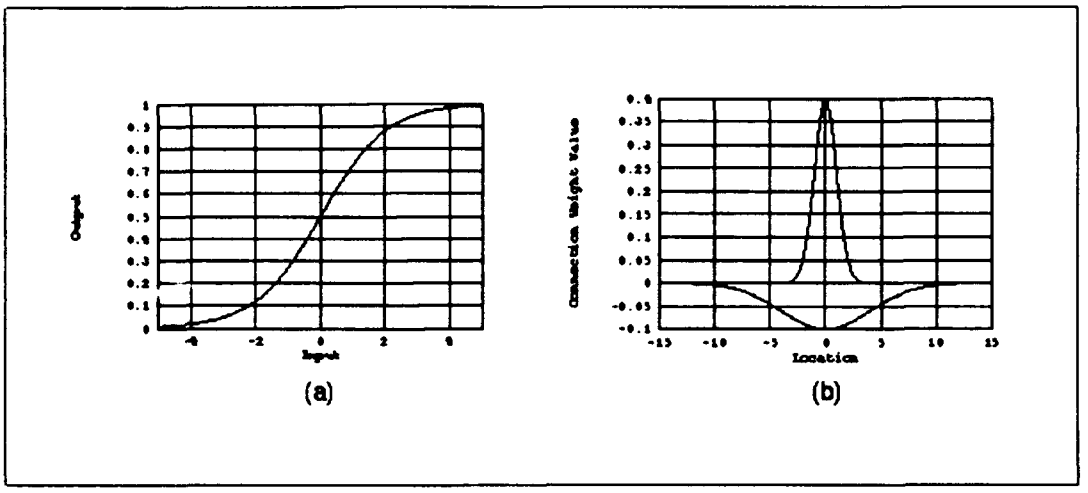

Figure 2.7. The sigmoid function is plotted in (a) while the gaussian connection weights are plotted in (b). In the gaussian plot, the narrow, positive curve represents an excitatory kernel and the wide, negative curve represents an inhibitory kernel.

positive and negative gaussians. The resulting equation would look like [Grossberg, 1987:7];

$$\frac{dx_i}{dt} = -Ax_i + \sum_{j=1}^{n} F_j(x_j)z_{ji} + I_i \tag{2.3}$$

where $F_j = f_j B_{ji} - g_j C_{ji}$.

The shunting dynamics of individual neurons is better modeled using a bounded STM equation that relies on automatic gain control. Grossberg gives the shunting STM equation as;

$$\frac{dx_i}{dt} = -Ax_i + (B_i - x_i)(\sum_{j=1}^{n} f_j(x_j)B_{ji}z_{ji}^{(+)} + I_i) - (x_i + D_i)(\sum_{j=1}^{n} g_j(x_j)C_{ji}z_{ji}^{(-)} + J_i) \tag{2.4}$$

Equation 2.4 makes use of a balanced positive and negative feedback signal over the interval $[B_i, -D_i]$. This is accomplished by multiplying the feedback signals by the difference between the current STM trace and the respective bound. As the trace approaches the upper bound, the inhibitory signals are given more weight to maintain the signal within the interval. Similarly, with a trace approaching the lower (negative) bound, the excitatory signal is weighted more to pull the

trace up into the bounded interval. The term I_i and J_i are inputs to the various cells i in the excitatory and inhibitory regions.

We would like to point out why the equation gets the name *shunting*. The response of a neuron is dependent on the amount of depolarization or hyperpolarization caused by the sum of all the synaptic junctions along the nerve's dendritic tree. The short-term trace, x, can be thought of as the level of ionic charge in the intercellular fluid. The shunting short-term trace equation explains how this charge changes. We must understand that B is the highest we would like the charge to be (although there are circumstances where this value can be exceeded). At any given time, an excitatory signal may cause an influx of positive ions into the cell. The amount of positive ions that cross the post-synaptic membrane depends on the difference between the current level of charge in the cell and the charge maximum $(B_i - x_i)$ as well as the strength of the excitation signal $(\sum_{j=1}^{n} f_j(x_j)B_{ji}z_{ji}^{(+)})$. If the intercellular fluid is too positively charged, the potential gradient will repel more positive ions at the channel.

As for the inhibitory signal, the same explanation holds but for $-D$ as the lowest we would like the charge to be. At the time an inhibitory signal reaches the synapse, the amount of negative ions flowing into the cell depends on the difference between the minimum charge and the current level of charge in the cell $(-D_i - x_i)$ and the inhibitory signal strength $(\sum_{j=1}^{n} g_j(x_j)C_{ji}z_{ji}^{(-)})$. In the next section we define z_{ji}.

2.3.2 The Long-Term Trace Equations. The long-term memory (LTM) traces contribute to the dynamics of the STM traces. We begin by looking at the Passive-Decay LTM equation. As its name implies, the trace decays at a constant rate of $-F_{ij}$ shown in Equation 2.5. This equation has a second term, $G_{ij}f_i(x_i)h_j(x_j)$, which is the Hebbian, nonlinear, learning term. This is a combination of a nonlinear sigmoid, $f_i(x_i)$, and a nonlinear learning function, $h_j(x_j)$.

$$\frac{dz_{ij}}{dt} = -F_{ij}z_{ij} + G_{ij}f_i(x_i)h_j(x_j) \tag{2.5}$$

A variant to the passive decay LTM equation is the Gated Decay LTM equation. In the gated decay version (Equation 2.6), the decay is not constant but is gated (multiplied) by a nonlinear learning function.

$$\frac{dz_{ij}}{dt} = h_j(x_j)(-F_{ij}z_{ij} + G_{ij}f_i(x_i)) \tag{2.6}$$

The equations described in this section are generic and must be tailored to any specific application. One such application is Öğmen and Gagné's model of the fly's visual system.

2.4 Development of a Motion Model.

The model presented by Öğmen and Gagné uses the Grossberg equations for short-term and long-term memory to explain experimental data from the fly's eye. Unlike the more general phenomenological models which treat the visual system as a black box, the neural model synthesizes the behavior of each biological element in order to mimic the whole system. Öğmen and Gagné's paper describes "directionally selective motion detection in the visual system of the fly" [Öğmen, 1990:488].

In the first part of their paper, Öğmen and Gagné modified Grossberg's equation for long-term memory to characterize a long term adaptation of a cell to it's input. The resulting equation is much simpler than Grossberg's and describes the accumulation of neurotransmitter in the vesicles of the synaptic button. Therefore, this equation is known as the neurotransmitter equation and is of the form;

$$\frac{dz_i}{dt} = \alpha(\beta - z_i) - (I + J_i)z_i \tag{2.7}$$

In the general case, a single cell would be connected to many synapses. However, Öğmen and Gagné were interested in modeling the fly's motion perception which is linear in nature. They conceived a simple model which they called the gated dipole (Figure 2.8).This model uses two

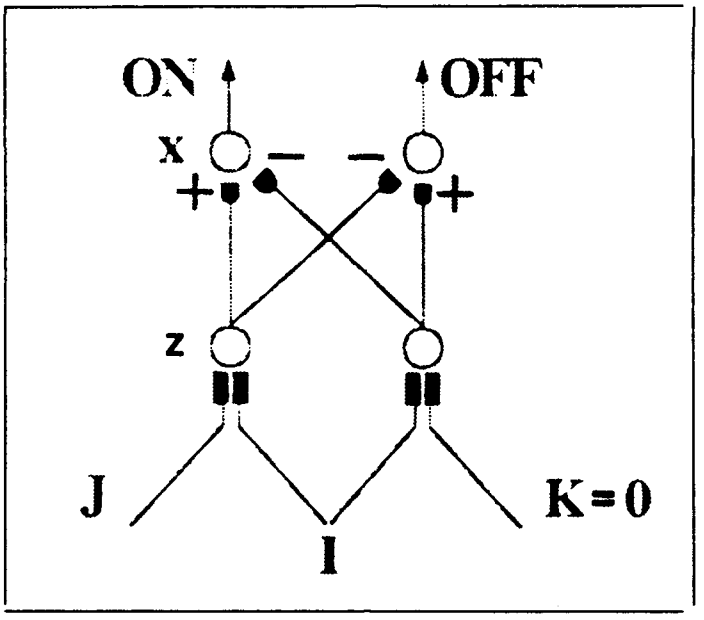

Figure 2.8. The Gated Dipole [Öğmen, 1990:492]

channels, the on-channel and the off-channel. Each channel was fed by it's own input times the channel's neurotransmitter level (z_{on} or z_{off}) as well as the opposing channel's input times that channel's neurotransmitter level (z_{off} or z_{on}). The cell's response was given by a second differential equation [Öğmen, 1990:491];

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)[I + J_i]z_i - (D + x_i)\sum_{k \neq i}[I + J_k]z_k \tag{2.8}$$

These equations will be examined in greater detail in Chapter III. For now, simply accept them as is. The gated dipole responds to a difference in the inputs of the respective synaptic junctions (z). In Figure 2.8, I represents an arousal signal (a signal common to all inputs), while J and K represent variations in inputs to either the on- or off-channel respectively.

If the on-channel is stimulated ($J \neq 0$) and the off-channel remains constant ($K = 0$), the gated dipole responds with an overshoot by the on-channel and an undershoot by the off-channel. The input synapse for the on-channel will eventually adapt to the new input level and cause both the on- and off-channels to return to a new, stable output level called a *plateau*. When the

stimulus is removed ($J = 0$), then the off-channel overshoots and the on-channel undershoots until the on-channel's synapse can readapt to the input. This behavior is called *antagonistic rebound* [Öğmen, 1990:491]. Both outputs will then plateau at the original resting potential. This behavior was explained in Öğmen and Gagné's paper but no figure was given. Our software recreates this behavior and so we have included a sample of it in Figure 2.9. If we look at the gated dipole in

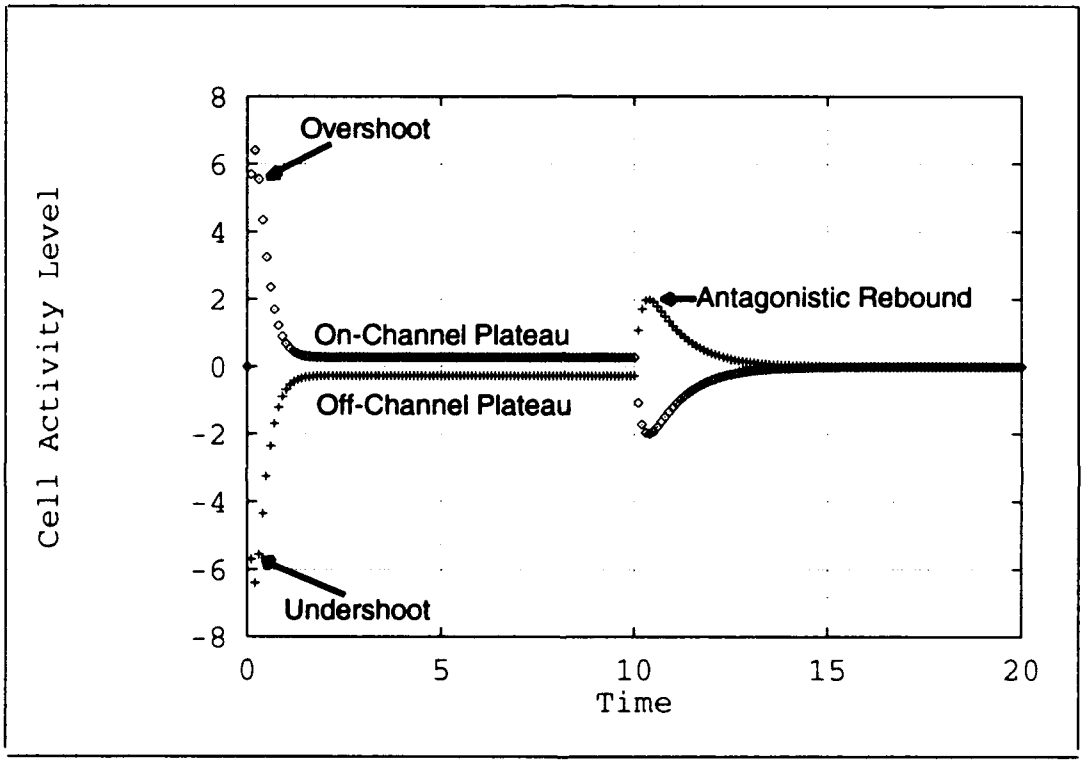

Figure 2.9. The Gated Dipole's Response. This figure shows the initial overshoot of the on-channel (diamonds) and undershoot of the off-channel (pluses) with the onset of a stimulus to the on-channel at time zero. As time continues, the two channels equilibrate to their respective steady states. A result of the on-channel's synaptic junction adapting to the input. Later, at time 10, the stimulus is removed and the on-channel undershoots while the off-channel overshoots. Both return to the original resting potential as the on-channel readapts.

relation to the receptive field and shunt-inhibitory synapses, we see that this structure has both an excitatory region and an inhibitory region of one. The strength of the excitation or inhibition signals are weighted by the value of the long-term trace and the shunt operates over the interval of $[B, -D]$.

After defining the simple gated dipole, Öğmen and Gagné proposed to create a linear array of dipoles as shown in Figure 2.10. This array is a one-dimensional structure with each gated dipole

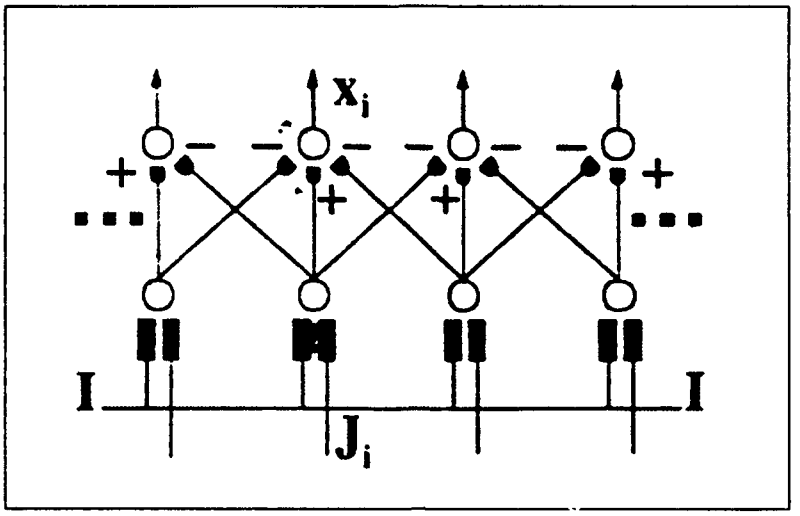

Figure 2.10. The Sustained Units [Öğmen, 1990:491]. This array of gated dipoles differ slightly from the primitive gated dipole in that the inhibitory region is now the two nearest neighbors while excitatory region remains one.

now having an inhibitory region of two (the two nearest neighbor dipoles). Because the activity, x_i, adapts to a constant stimulus, the model is called a *sustained unit.* The next evolutionary step in this model is the two-dimensional form. But first, Öğmen and Gagné discuss recurrent and nonrecurrent architectures for delaying the inhibitory signal.

2.4.1 Recurrent and Nonrecurrent Models. In Equations 2.8 and 2.7, the inhibitory and excitatory signals arrive at the synaptic junction simultaneously. Earlier we stated that directional selectivity depends on different propagation times between inhibitive and excitatory signals. Öğmen and Gagné addresses two methods of delaying a signal - recurrent and nonrecurrent. Figure 2.11 illustrates the difference in delay methods using a wiring diagram. The linear, recurrent model (left) can be characterized by the following equation;

$$x(t) = I(t) - x(t - \Delta t) \tag{2.9}$$

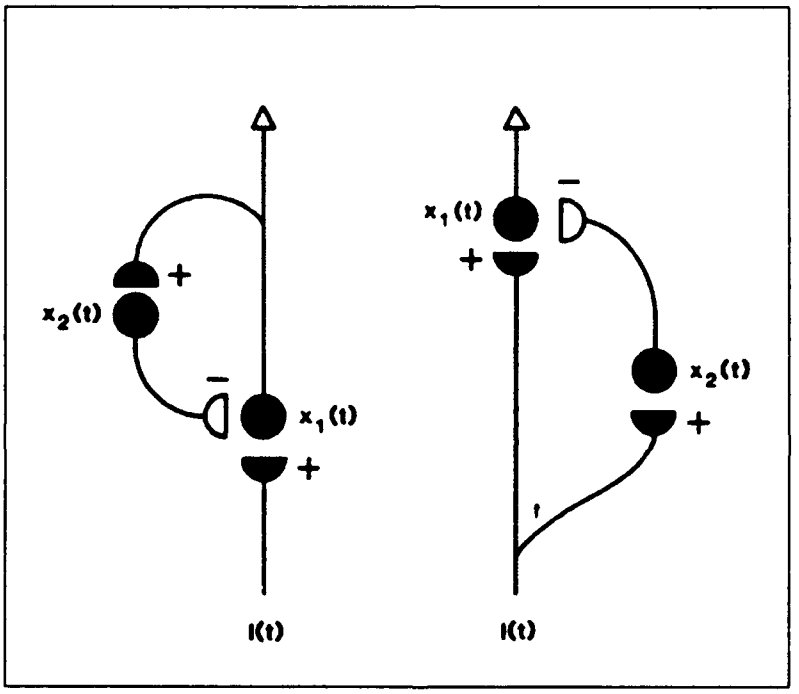

Figure 2.11. Recurrent (left) and Nonrecurrent (right) signal paths. The recurrent path feeds a delayed output of the cell back into itself and has definite problems with oscillations given a constant input. The nonrecurrent path is a feed forward delay and has no such problems. [Öğmen, 1990:497]

Equation 2.9 has two imaginary poles with zero real parts indicating a marginally stable system. In other words, the system is a pure harmonic oscillator [Reid 1983, 65]. The nonrecurrent anatomy delays the input signal which the neuron uses as an inhibitory signal to temporally dampen the output of the neuron. Öğmen and Gagné generalize these linear models to nonlinear neural models and found that the recurrent version still oscillates given the right conditions. Therefore, the nonrecurrent form is preferred.

The nonrecurrent equation for spacial and temporal inputs is given as;

$$\begin{aligned}\frac{\partial x(s,t)}{\partial t} &= -Ax(s,t) + (B - x(s,t))\,(u_e(s,t) * G_e(s)) \\ &\quad - (D + x(s,t))\,(u_i(s,t) * G_i(s)) \end{aligned} \tag{2.10}$$

where the $*$ is the convolution operator, s is the spacial variable and G_e and G_i describe the excitatory and inhibitory receptive fields (spacial kernels) with respective inputs u_e and u_i. The difference between this equation and Equation 2.8 is the spacial operator is continuous and the receptive fields are far more complicated. The excitatory kernel is chosen to be a narrow gaussian while the inhibitory kernel is a wide gaussian (Refer back to Figure 2.7).

2.4.2 Synchronous and Asynchronous Inputs. If $u_e(s,t) = u_i(s,t)$ then the system is called *synchronous.* However, as was stated earlier, directional selectivity depends on the inhibitory signal propagation speed being slower than the excitatory signal. Therefore, if $u_e(s,t) = u_i(s,t-\tau)$ then the system is ***asynchronous*** by way of a delayed input to the inhibitory region. In the synchronous case, Equation 2.10 can be simplified to;

$$\frac{\partial x(s,t)}{\partial t} = -C_1(s,t)x(s,t) + C_2(s,t) \tag{2.11}$$

The spacial kernels can be thought of as spacial weights on the input signals ($\tilde{u}(s,t)$). To keep the function from varying greatly due to various signal power distributions, the kernels must

be able to normalize the input signal. This is done by defining a background activity ($U(t)$) as the total power of the signal (Equation 2.12) and the sum of the weights as unity (Equation 2.13).

$$U(t) = \int_s u(s,t)ds \tag{2.12}$$

$$\int_s \tilde{u}(s,t)ds = 1 \tag{2.13}$$

The input signal can then be expressed as;

$$u(s,t) = \tilde{u}(s,t)U(t) \tag{2.14}$$

By representing the input signal in this way, we can quickly identify the power of the signal and remove it from our calculations by dividing by $U(t)$. This is called discounting the illuminance.[3] Replacing $u(s,t)$ with the new form, the coefficients on Equation 2.11 become;

$$C_1(s,t) = A + U(t)(\tilde{u}(s,t) * SOG) \tag{2.15}$$

$$C_2(s,t) = VU(t)(\tilde{u}(s,t) * DOG) \tag{2.16}$$

where the Sum of Gaussians is $SOG = G_e(s) + G_i(s)$ (Figure A.10) and the difference of gaussians is $DOG = G_e(s) - G_i(s)$ [Öğmen 1990:500]. Here $B = D = V$ for simplicity. The asynchronous version of Equation 2.10 is identical in form to Equation 2.11 but with more complicated coefficients.

$$C_1(s,t) = A + (U(t)\tilde{u}(s,t) * G_e(s) + U(t-\tau)\tilde{u}(s,t-\tau) * G_i(s)) \tag{2.17}$$

$$C_2(s,t) = V\,(U(t)\tilde{u}(s,t) * G_e(s) - U(t-\tau)\tilde{u}(s,t-\tau) * G_i(s)) \tag{2.18}$$

Using the synchronous and asynchronous models, Öğmen and Gagné illustrated the one-dimensional response to moving a white bar on a black background (Figure 2.12). In Figure 2.13

[3]Carver Mead and Misha Mahowald model this early vision effect by providing a horizontal network of resistors which determines the power of the signal in the receptive field in logarithmic form. The bipolar element then subtracts it from the logarithmic representation of the input to form an output which has the illuminance discounted [Mahowald, 1991:79] (See Appendix A page 21).

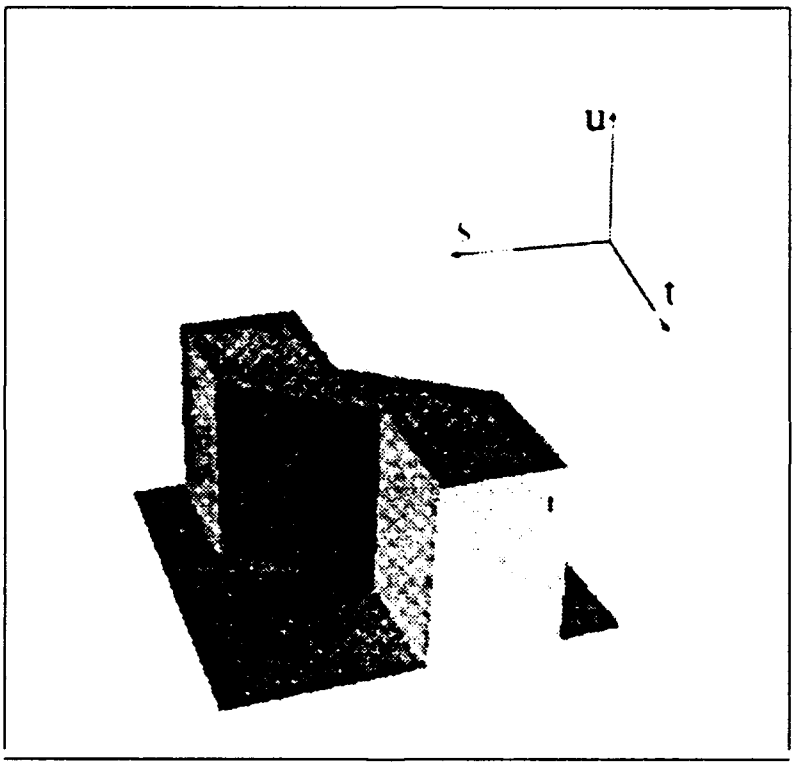

Figure 2.12. Input to the synchronous and asynchronous models developed by Öğmen and Gagné. The three dimensions represent spacial, temporal, and intensity level [Öğmen, 1990:499].

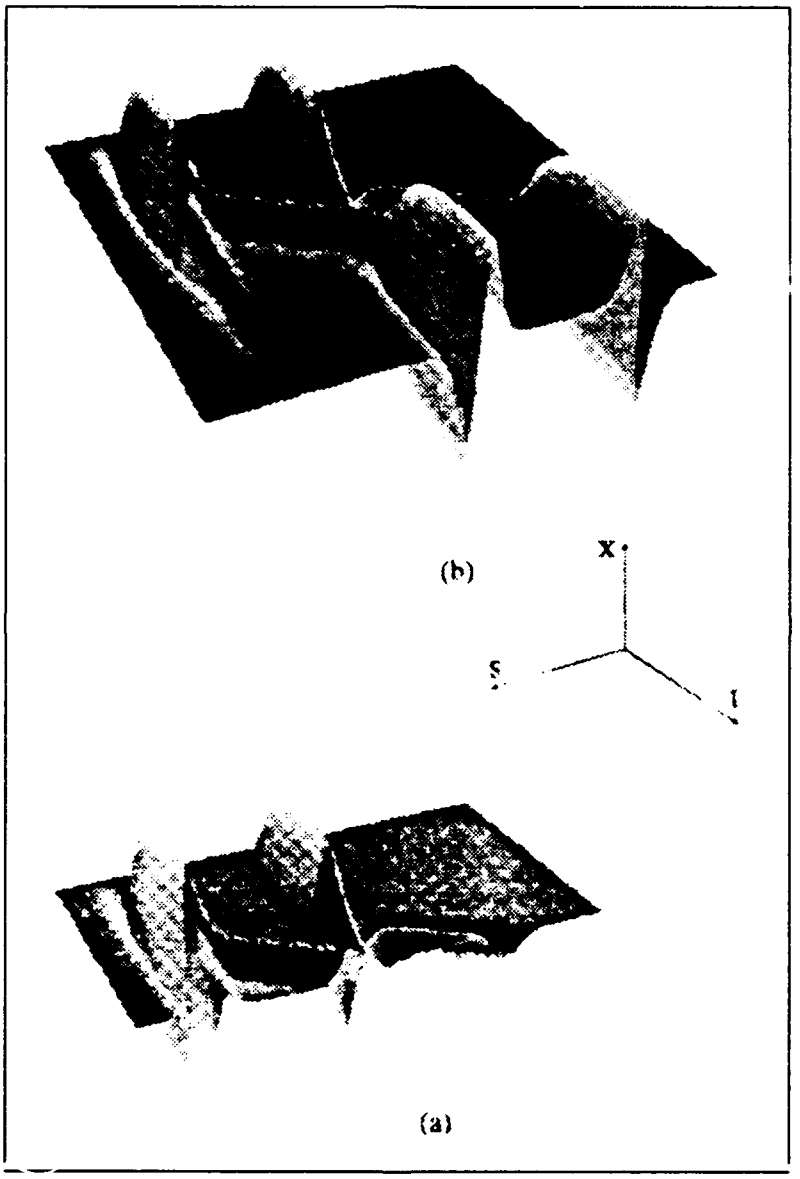

Figure 2.13. Response of the synchronous model to a moving input. Here the model output is plotted against spacial and temporal axis. The full temporal result is given above and a snapshot of the motion is shown below.[Öğmen, 1990:492]

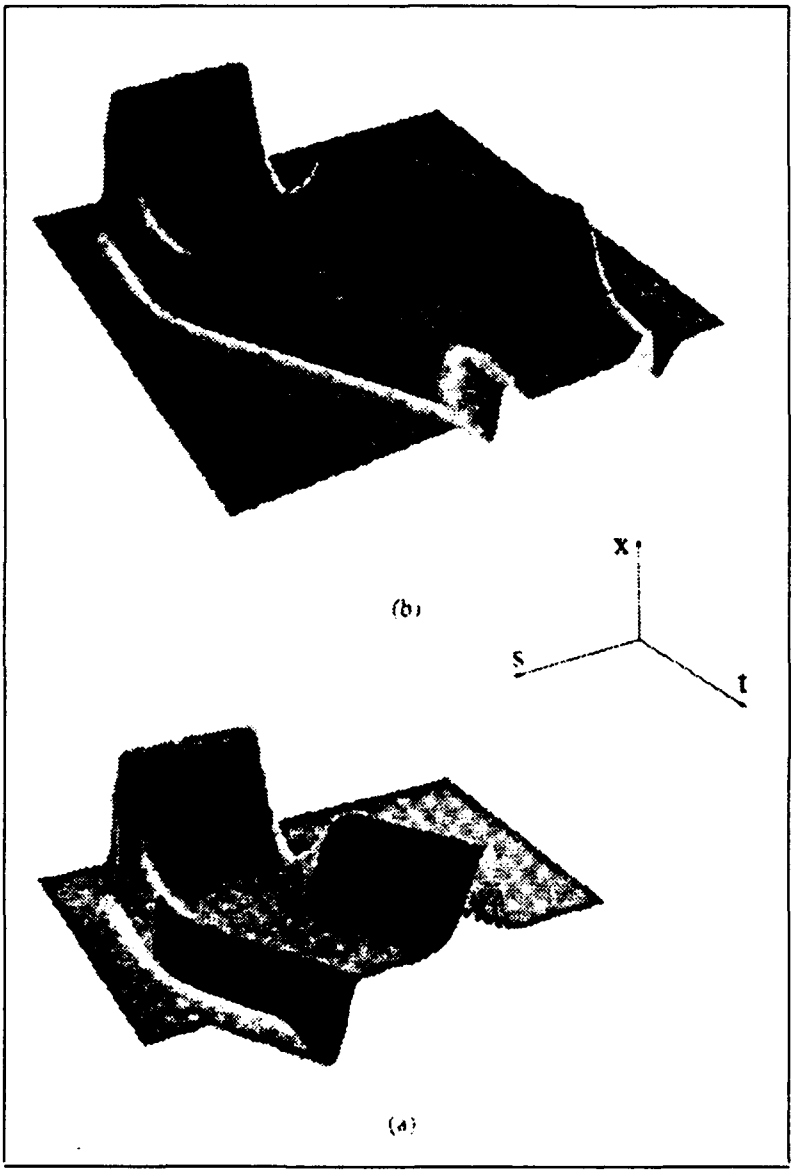

Figure 2.14. Response of the asynchronous model to a moving input. Here the model output is plotted against spacial and temporal axis. The full temporal result is given above and a snapshot of the motion is shown below[Öğmen, 1990:500]

we see that the synchronous model slowly responds to the onset of an input and responds more vigorously to stationary, sustained inputs. When the input moves, the model responds at a lower level but does indeed track the input. In Figure 2.14, the asynchronous response seems to highlight transient inputs more than stationary ones. Therefore, the two models trade-off sensitivity to motion for sensitivity to stationary inputs. The asynchronous model responds to leading edges with a positive going *wave* in the direction of motion or in front of the object while trailing edges have a negative going wave to the rear of the object. In the synchronous model, the difference between leading and trailing edges is not so distinct.

Having developed the sustained and on-off models early in their paper, Öğmen and Gagné fail to introduce an asynchronous model driven by inputs characterized by the neurotransmitter equations. We believe that this is an important area for research as it may well provide a temporal adaptation of inputs and further define a model sensitive only to motion in the input and insensitive to stationary inputs (following a transient adaptation period).

2.5 Conclusion

This chapter has taken us from basic concepts in the biology of the retina through the theoretical background based in differential equations and finally on to the refined model of a cell sensitive to motion. More specifically, we have;

- defined the terms -
  - *Receptive Field* consists of a central region and an annular surround. It can cause the central neuron to react positively to an illuminated center and dark surround (on-center off-surround) or vise versa (off-center on-surround).
  - *Shunt Inhibition* is a form of inhibition that effects the post-synaptic neuron only in the presence of an excitatory signal.

- *Directional Selectivity* describes a ganglion which responds to motion in a preferred direction.

- looked at the basic mathematical theory as defined by Stephen Grossberg for shunting short-term memory and long-term memory traces. These equations form the basis for our model and are very general.
- reviewed the most recent article by Öğmen and Gagné which developed the gated-dipole model and the sustained unit as primitive building blocks for further work. These blocks are then extended into two dimensions and given the ability to delay the inhibitory signal.
- postulated an area overlooked by Öğmen and Gagné. That is, adding long-term memory to their two-dimensional, asynchronous model.

With a firm understanding of recent work made in the area of visual processing, we can turn to application. In Chapter III we will look at the model theory to a greater degree of detail than given in this chapter. Since the modeling software was designed, coded, and tested as a parallel thesis effort, we only look at the architecture and results of simulations designed to validate the model. In Chapter III, we will look at some basic model responses to one-dimensional input patterns and examine the results of post processing used to enhance the model's transient behavior. At the close the next chapter, the reader will be ready to investigate the model's response to two-dimensional image sequences.

III. Modeling the Spacial and Temporal Response of the Receptive Field

In Chapter II we saw that of the retina the amacrine and ganglion neurons depend on a receptive field for motion enhancement. Motion is identified by using a shunt inhibitory region rather than natural inhibition. In this chapter we look again at the equations developed by Grossberg [Grossberg, 1987:6] and tailored by Öğmen and Gagné [Öğmen, 1990:491] to characterize the motion sensitive neural structure. We must solve the neurotransmitter equation and the cell-activity equation in order to understand what assumptions were made in order to simplify them and predict their behavior.

The software used to simulate these models was tested at various stages of development. The results of these test not only validate our program but provide graphic explanation of the model's response to various inputs. The design and validation of the software is not the subject of this thesis. We will define the model's architecture suggest different post processing techniques which will be used in Chapter IV in the two-dimensional simulations. We begin with the neurotransmitter model.

3.1 The Neurotransmitter Model

In this section we analyze the differential equation Öğmen and Gagné used to characterize their model's first order node. They chose to call this a neurotransmitter equation because of the biological analogy of this node to the synapse. Before we tear into this equation, lets look at the mechanism in the synapse that this equation models. As discussed in Appendix A, the presynaptic membrane encloses many vesicles carrying neurotransmitter chemicals. When the presynaptic cell's potential rises, vesicles fuse with the membrane and discharge their contents into the synaptic cleft [Stevens, 1979:16] (Figure 3.1). The amount of transmitter released is directly proportional to the size of the signal received at the synaptic junction and the level of transmitter available to the

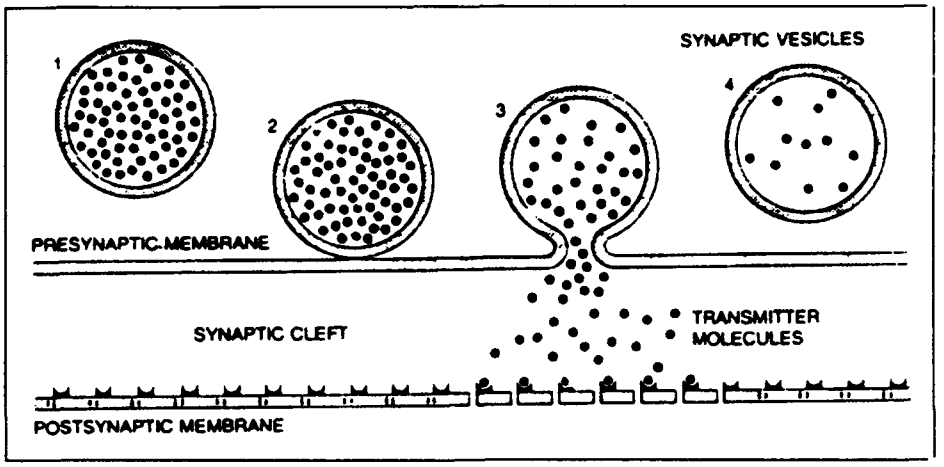

Figure 3.1. The process of exocytosis. A rise in potential at the presynaptic membrane causes vesicles to fuse to the membrane and discharge their contents into the cleft. They are then reclaimed, reformed, and refilled with transmitter [Stevens, 1979:16]

synapse. Therefore, the amount of transmitter in each vesicle determines the ability of the synapse to transmit a signal to the target cell.

Öğmen and Gagné mathematically characterized this process using a first-order differential equation of the form [Öğmen, 1990:491];

$$\frac{dz_i}{dt} = \alpha(\beta - z_i) - (I + J_i)z_i \tag{3.1}$$

Here, α is the spontaneous decay of the amount of neurotransmitter in the synapse, z_i, and β is the resting level of neurotransmitter in the synapse. The variable I is the normal[1] intensity and J_i is the difference[2] in intensity from I at cell i. We will solve Equation 3.1 in the next section.

3.1.1 Theoretical Analysis. In their paper, Öğmen and Gagné provided only the solution to Equation 3.1 and not the derivation. We decided to derive the solution in order to better understand the model. In so doing, we found that they made certain assumptions and simplifications which are important to understand before we proceed. In the following derivation we replaced $(I(t) + J_i(t))$

[1] I is a local norm. The function $I(x, y)$ represents a smooth intensity curve over the special region of the cells.
[2] If I is a smoothed curve, then J represents spikes on the curve at each input.

with $\bar{I}(t)$ to shorten the length of these equations. We begin by placing the equation in a more standard form.

$$\frac{dz(t)}{dt} = \alpha(\beta - z(t)) - \bar{I}(t)z(t) \tag{3.2}$$

$$\frac{dz(t)}{dt} + [\alpha + \bar{I}(t)]z(t) = \alpha\beta \tag{3.3}$$

Multiplying by an integrating factor of $e^{\int_0^t \sigma(y)dy}$ where $\sigma = \alpha + \bar{I}(t)$ we get;

$$e^{\int_0^t \sigma(y)dy}\left(\frac{dz(t)}{dt} + [\alpha + \bar{I}(t)]z(t)\right) = e^{\int_0^t \sigma(y)dy}(\alpha\beta) \tag{3.4}$$

$$\frac{dz(t)}{dt}\left(e^{\int_0^t \sigma(y)dy}z(t)\right) = e^{\int_0^t \sigma(y)dy}(\alpha\beta) \tag{3.5}$$

Integrate from 0 to t:

$$\left[e^{\int_0^t \sigma(y)dy}z(t)\right]\Big|_0^t = \int_0^t e^{\int_0^\tau \sigma(y)dy}(\alpha\beta)d\tau \tag{3.6}$$

$$e^{\int_0^t \sigma(y)dy}z(t) - z(0) = \int_0^t e^{\int_0^\tau \sigma(y)dy}(\alpha\beta)d\tau \tag{3.7}$$

$$z(t) = e^{-\int_0^t \sigma(y)dy}\left(z(0) + \int_0^t e^{\int_0^\tau \sigma(y)dy}(\alpha\beta)d\tau\right) \tag{3.8}$$

If $\bar{I}(t) = 0$ for $t < 0$ and $\bar{I}(t) = I$ (a constant) for $t \geq 0$, then $z(0^-) = \beta$ and $\sigma(y) = (\alpha + I)$ which is also a constant.

$$z(t) = e^{-(\alpha+I)t}\left[\beta + \int_0^t e^{(\alpha+I)\tau}(\alpha\beta)d\tau\right] \tag{3.9}$$

$$= e^{-(\alpha+I)t}\left[\beta + \alpha\beta\left(\frac{1}{\alpha+I}(e^{(\alpha+I)\tau})\Big|_0^t\right)\right] \tag{3.10}$$

$$= e^{-(\alpha+I)t}\left[\beta + \alpha\beta\left(\frac{1}{\alpha+I}(e^{(\alpha+I)t} - 1)\right)\right] \tag{3.11}$$

$$= \beta e^{-(\alpha+I)t} + \frac{\alpha\beta}{\alpha+I}\left(e^{-(\alpha+I)t}e^{(\alpha+I)t} - e^{-(\alpha+I)t}\right) \tag{3.12}$$

$$= \frac{\alpha\beta}{\alpha+I}\left(1 - e^{-(\alpha+I)t}\right) + \beta e^{-(\alpha+I)t} \tag{3.13}$$

This is the solution given by Öğmen and Gagné. However, it is important to know that this assumed a step input. Therefore, this is the step response of the system and as Öğmen and Gagné pointed out, the steady-state value for this equation would be $z_{ss} = \frac{\alpha\beta}{\alpha+I}$. At this point we must recognize that the equation describes a linear, time-variant system. If we allow for a nonzero initial condition, say I_{t_0}, then the resulting solution would be;

$$z(t) = \frac{\alpha\beta}{\alpha + I}\left(1 - e^{-(\alpha+I_{t_0})t}\right) + \frac{\alpha\beta}{\alpha + I_{t_0}} e^{-(\alpha+I_{t_0})t} \tag{3.14}$$

We see that $I_{t_0} = 0$ is a special case of Equation 3.14 giving us Equation 3.13. As the input changes, it is important to see that a solution for $z(t)$ can only be obtained in a piecewise fashion if we were to use Equation 3.14. An alternative is to build a state model which is propagated using some numerical technique. The advantage of having Equations 3.13 and 3.14 is in validating the software model built from Equation 3.1.

3.1.2 Neurotransmitter Response. Figure 3.2 illustrates the response of the neurotransmitter node given a series of step changes as input. In this figure, the model was initially forced to steady-state given an input of 1.0. From the graphs of Figure 3.2, we see that the coefficient α effects fall-time and rise-time of the system's response. As would be expected, a decrease in α causes a slower model response than the higher value.

In the next section we will see that the inputs to the cell-activity model is always the current input to the neurotransmitter model ($I(t)$) times the current neurotransmitter level ($z(t)$). Therefore, we plotted the product of $I(t)$ and $z(t)$ in Figure 3.3. The form of this plot is exactly what Öğmen and Gagné describe in their article. The output overshoots to a value of $\frac{\beta \times I}{\alpha \times I_{t_0}}$ and eventually comes to rest at the steady-state value multiplied by the input ($z_{ss} \times I$). These values are found using Equation 3.14 with $t = 0$ and $t = \infty$ respectively. Next we look at the cell-activity model.

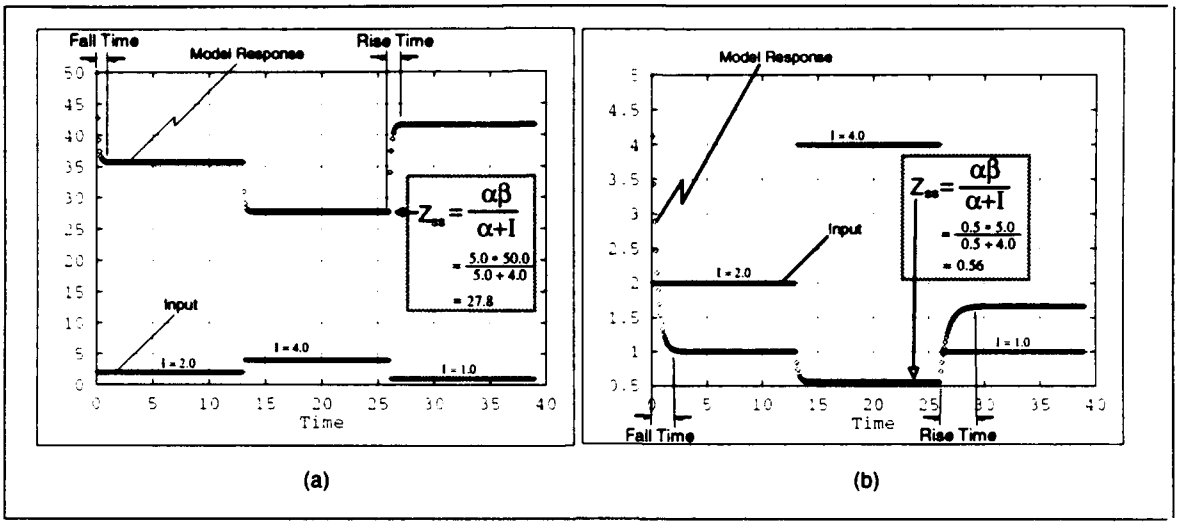

Figure 3.2. Response of the neurotransmitter model to a series of step changes in input. Plots (a) and (b) differs from one another only the values used for the natural decay term (α) and the resting level (β). In (a) $\alpha = 5.0$ and $\beta = 50.0$. In (b) $\alpha = 0.5$ and $\beta = 5.0$. The plots show the model response to changing input values over time. Each input is held long enough for the model to reach steady-state. The difference in fall-time or rise-time is indicated to show the effect of changing α. Plot (b) shows a slower response due to the smaller value of α.

3.2 The Cell-Activity Model

The biological neuron receives information from neighboring cells through synaptic junctions in its dendritic tree. Öğmen and Gagné's equation for a cell's activity characterizes the internal cell potential due to a temporal and spacial distribution of inputs. This internal potential is called a slow potential [Levine, 1985:89]. In a normal ganglion, the cell produces an output spike or action potential only after the cell potential reaches some threshold value [Levine, 1985:90]. The cell is able to adapt to inputs by varying this level as a function of the current activity level of the cell. In Öğmen and Gagné's cell-activity model, no account is made to develop action potentials. We accept this for now. Later we will look at post processing that simulates this behavior.

The cell-activity model characterizes the behavior of biological neurons by providing a natural decay of activity, an excitatory term, and an inhibitory term. Because the model was designed for the fly's visual perception, it was defined with an excitatory region and inhibitory region of one cell each (neighboring cells in the sustained unit - Section 2.4). We look at the general case where

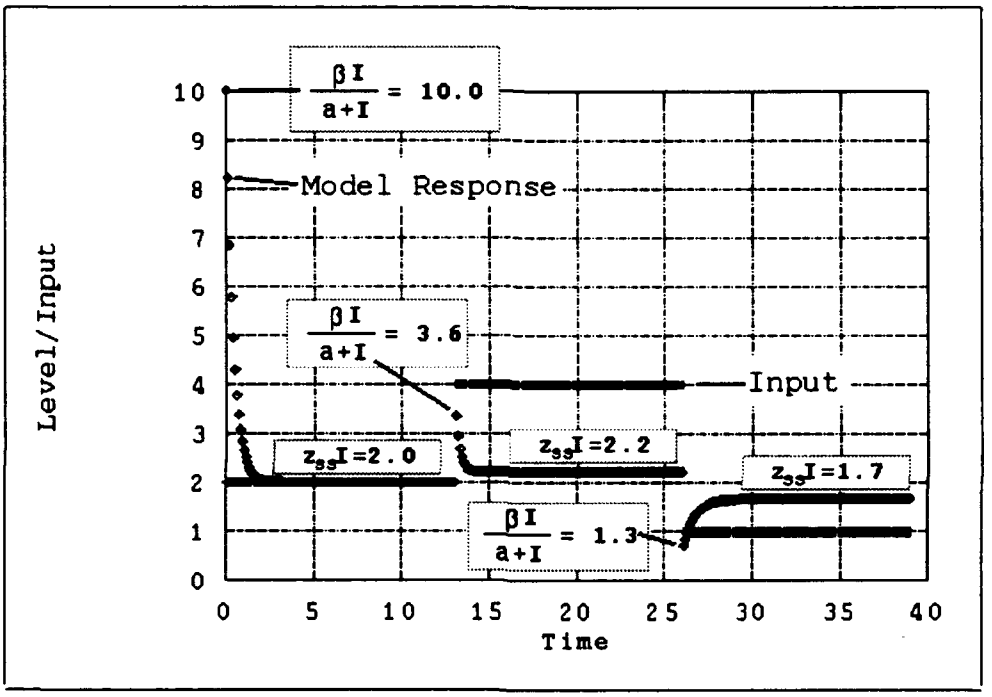

Figure 3.3. Response of the neurotransmitter model multiplied by the current input value $I(t)$. Here $\alpha = 0.5$ and $\beta = 5.0$. Diamonds represent the product of $z(t)$ and $I(t)$ while the pluses represent input values. The overshoot values follow the equation $\frac{\beta \times I}{\alpha \times I_{t_0}}$ and are 10.0, 3.6, and 1.3 for steps 2.0, 4.0, and 1.0 respectively. The plateau values correspond to $\frac{\beta \times I}{\alpha + I} \times I$ and are 2.0, 2.2, and 1.7 respectively.

these regions are of variable size and develop an understanding of the equation's solutions as we did with the neurotransmitter model. Equation 2.8 is rewritten here for convenience.

$$\frac{dx_i}{dt} = -Ax_i(t) + (B - x_i(t))[I + J_i]z_i - (D + x_i(t))\sum_{k\neq i}[I + J_k]z_k(t) \tag{3.15}$$

It can be solved in the same fashion as Equation 3.1 but must be rearranged to identify the time-varying coefficients. We replaced $[I + J_i]$ with $\bar{I}$ to shorten the equation's length.

$$\frac{dx_i}{dt} = \left[(-A - \bar{I}_i z_i(t)) - \sum_{k\neq i}\bar{I}_k z_k(t)\right] x_i(t) + \left[B\bar{I}_i z_i(t) - D\sum_{k\neq i}\bar{I}_k z_k(t)\right] \tag{3.16}$$

The first coefficient has three terms. The second two represent a summation of all $\bar{I}(t)z(t)$ products in the receptive field and can be combined to form a single summation. The second coefficient could also be combined in this manner if $B = -D$, an assumption that will be made later. But for now, we maintain that B and D can take on any value. Equation 3.16 can be simplified by making the following substitutions;

$$\Phi(t) = (-A - \bar{I}_i z_i(t)) - \sum_{k\neq i}\bar{I}_k z_k(t) \tag{3.17}$$

$$= \left(-A - \sum_i \bar{I}_i z_i(t)\right) \tag{3.18}$$

$$\Psi(t) = B\bar{I}_i z_i(t) - D\sum_{k\neq i}\bar{I}_k z_k(t) \tag{3.19}$$

Then Equation 3.16 can be written as;

$$\frac{dx_i}{dt} = \Phi(t)x_i(t) + \Psi(t) \tag{3.20}$$

As before, we introduce an integrating factor $e^{-\int_0^t \Phi(y)dy}$.

$$e^{-\int_0^t \Phi(y)dy}\left(\frac{dx_i(t)}{dt} - \Phi(t)x_i(t)\right) = e^{-\epsilon\int_0^t \Phi(y)dy}\Psi(t) \tag{3.21}$$

The steps in solving this equation are identical to those used in solving the neurotransmitter equation. The final answer turns out to be;

$$x_i(t) = e^{\int_0^t \Phi(y)dy}\left(x_i(0) + \int_0^t e^{-\int_0^\tau \Phi(y)dy}\Psi(\tau)d\tau\right) \tag{3.22}$$

As was pointed out by Öğmen and Gagné, the steady-state value for this equation is given below. Here we must reintroduce the full form for $\bar{I}$. I is the initial value of the input and J_i is the step change to the input for the i^{th} cell.

$$x_{i(ss)} = \frac{-\Psi(t)}{\Phi(t)} \tag{3.23}$$

$$= \frac{B[I + J_i]z_{i(ss)} - D\sum_{k\neq i} Iz_{k(ss)}}{A + Iz_{i(ss)} + \sum_{k\neq i} Iz_{k(ss)}} \tag{3.24}$$

Although the software we build uses the simpler differential equation, it is important to understand where the system will settle in steady-state. We might also note that the cell-activity model is a linear, time-varying, first-order differential equation.

Because of the time-varying nature of the model, it is very difficult to predict the behavior of its response over time. In the simplified case of the gated dipole used by Öğmen and Gagné, the model can be analyzed using the steady-state versions of the differential equations for neurotransmitter and cell activity. These equations are reprinted below [Öğmen, 1990:491].

$$x_{on} = \frac{B[I + J_{on}]z_{on} - DIz_{off}}{A + [I + J_{on}]z_{on} + Iz_{off}} \tag{3.25}$$

$$x_{off} = \frac{BIz_{off} - D[I + J_{on}]z_{on}}{A + [I + J_{on}]z_{on} + Iz_{off}} \tag{3.26}$$

As Öğmen and Gagné explain the system's behavior, when the on-channel is stimulated with $J_{on} = J$ and $J_{off} = 0$, the system will respond with the on-channel overshooting and the off-channel undershooting. Since the dynamics of the neurotransmitter is slower than that of the cell-activity,

Table 3.1. Parameters used when testing the Amacrine object. The lower bound term is varied in the following discussion to illustrate its effect on the response curve of the soma.

α	0.5	Natural Decay Term
β	5.0	Target Value
A	5.0	Natural Decay Term
B	45.0	Upper Bound
D	Variable	Lower Bound

the on-channel neurotransmitter will eventually equilibrate to a value of;

$$z_{on} = \frac{\alpha\beta}{\alpha + I + J} \tag{3.27}$$

while the value of z_{off} will remain unaffected [Öğmen, 1990:491]. This will account for a decrease in the on-channel and off-channel activity level. Both will come to a plateau at some level given by Equations 3.25 and 3.26. When the input is removed ($J_{on} = 0$), then both channels will rebound with an overshoot or undershoot then plateau to the original steady-state values. Illustration of this behavior is given in Figure 3.4 (Table 3.1 gives the parameters used for in the simulations). This figure also shows the effect of changing the lower bound of the cell-activity model.

One final note before we move on to defining the model architecture. If a spacially homogeneous input is applied to the gated dipole ($J = K$) then Equations 3.25 and 3.26 become [Öğmen, 1990:491];

$$x_{on} = x_{off} = \frac{\alpha\beta(B - D)(I + J)}{A(\alpha + I + J) + 2\alpha\beta(I + J)} \tag{3.28}$$

From this equation we see that there is no response to a homogeneuos input. Illustration of a homogenous input response is given in Figure 3.5.

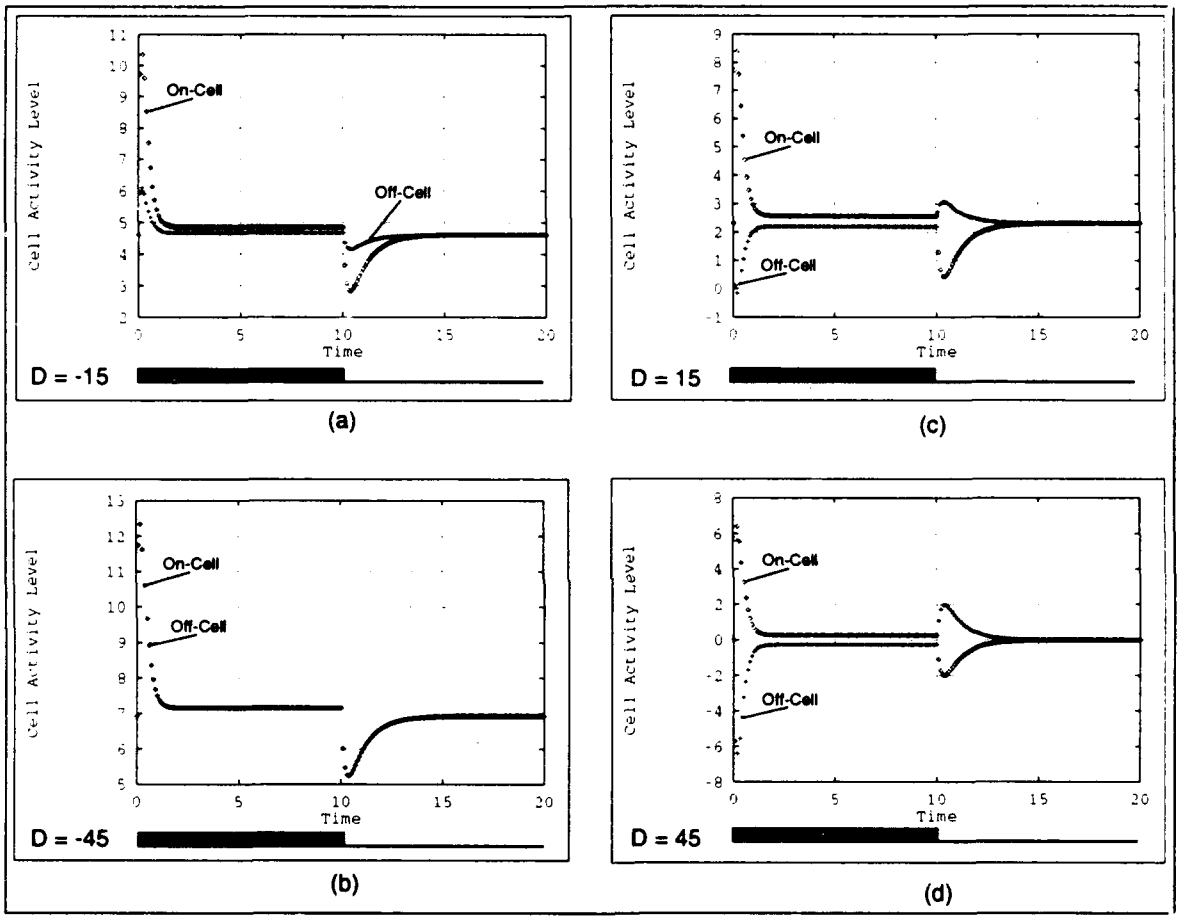

Figure 3.4. The four plots in this figure depict how a change in lower bound effects the response of the gated dipole. The input to these plots consists of illuminating the om-cell for a short time. After ten time units, the illumination is removed. The input is given as the solid black bars blow each figure. In plot (a), the lower bound is set at -15. In plot (b) the lower bound is set equal to the negative of the upper bound (-45). Here we see that setting the lower bound to a negative value results in eliminating the inhibitory region. In essence, we have created two excitatory regions. In plots (c) and (d) the lower bound is set at 15 and 45 respectively.

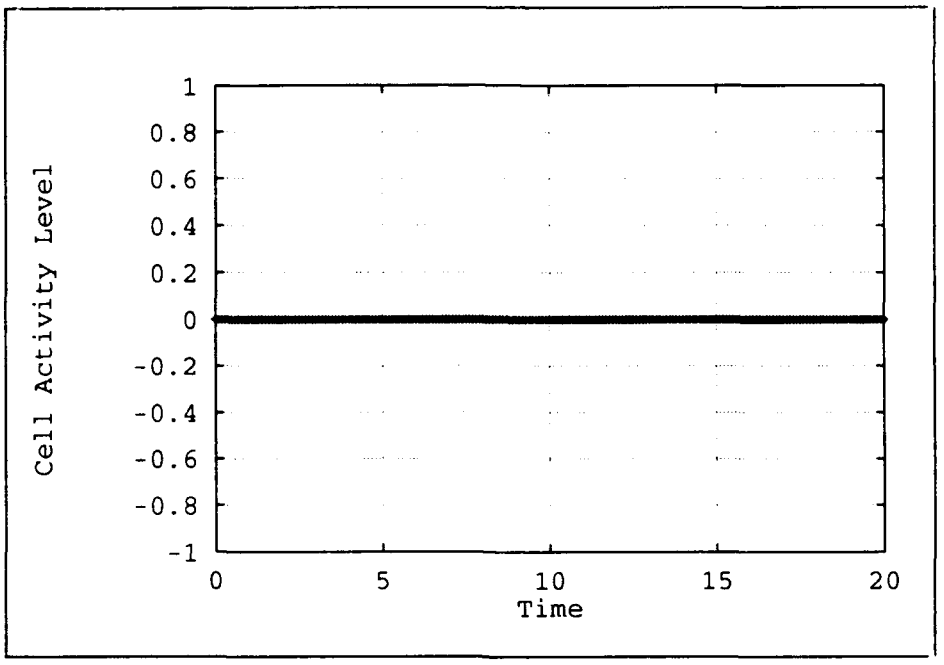

Figure 3.5. Response to a homogenous input with the parameter of $D = 45$. This figure show that the gated dipole has no response to homongeneuos inputs.

3.3 Model Architectures

The equations examined above represent the basic building blocks of our model. We found that the synchronous forms of the neural model respond favorably to stationary inputs while the asynchronous models are more sensitive to transient inputs. To explain the differences between the two models we compared we turn to a theoretical analysis of the spatio-temporal form of the cell-activity model. A conceptual view of the one- and two-dimensional form of the receptive field is given in Figures 3.6 and 3.7. Extension of the basic gated dipole to the spatio-temporal architectures pictured in these figures is given below.

3.3.1 Theoretical Analysis. The gated dipole was mathematically characterized by two differential equations given in Chapter II (reprinted here for convenience).

$$\frac{dx_{on}}{dt} = -Ax_{on} + (B - x_{on})[I + J_{on}]z_{on} - (D + x_{off})[I + J_{off}]z_{off} \tag{3.29}$$

$$\frac{dz_i}{dt} = \alpha(\beta - z_i) - (I + J_i)z_i \tag{3.30}$$

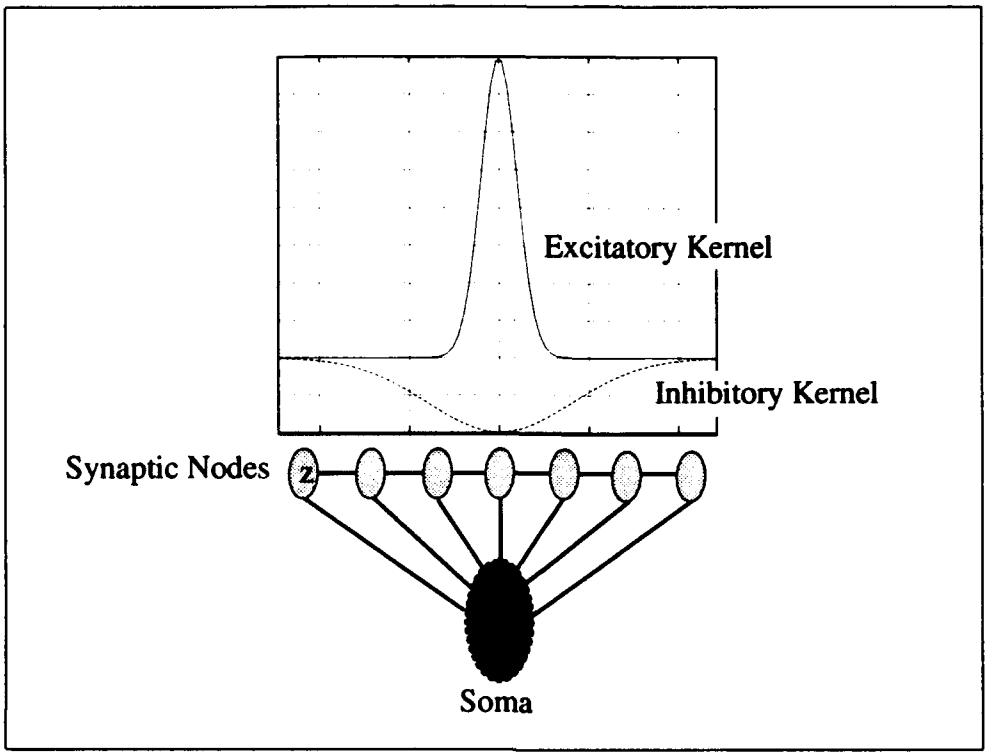

Figure 3.6. The simplified neuron model drawn in one dimension. This figure shows the continuous gaussian kernels above the synaptic inputs.

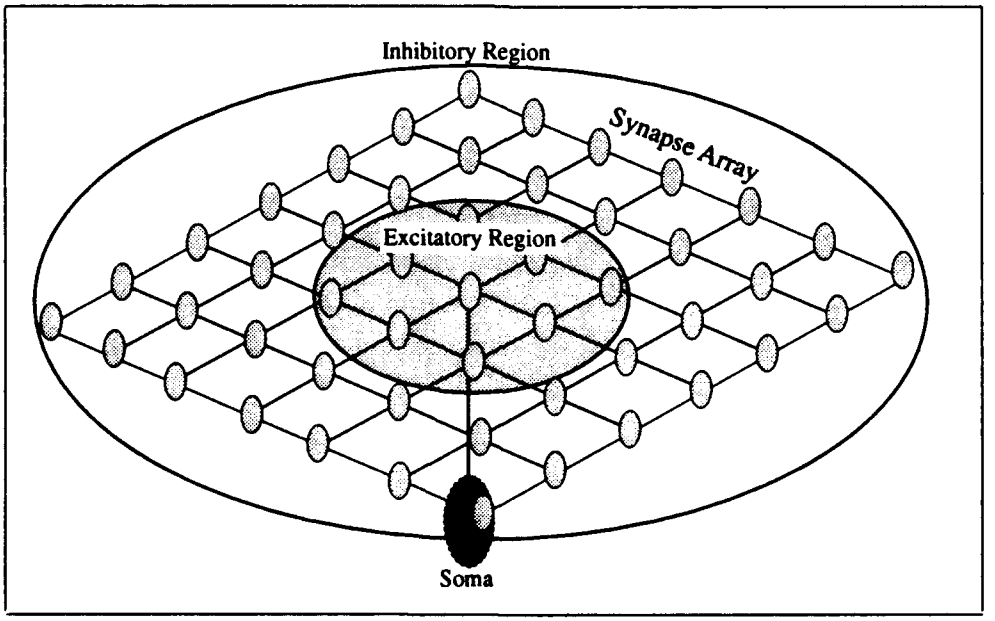

Figure 3.7. The neuron model drawn in two dimensions. Here there are many synaptic nodes supporting a single soma node.

For the most part, Equation 3.30 is not effected by the change from gated dipole to neuron model. It is Equation 3.29 that will be dramatically changed. Our software supports a model characterized by Equation 2.10 (also reprinted below for convenience).

$$\frac{\partial x(s,t)}{\partial t} = -Ax(s,t) + (B - x(s,t))(u_e(s,t) * G_e(s)) - (D + x(s,t))(u_i(s,t) * G_i(s)) \tag{3.31}$$

We mathematically accomplish the modification in two steps. The first step expands the receptive field without regard to connection weights. The mathematical representation of this intermediate step is similar to Equation 2.8 but allows for a larger excitatory region as indicated in Equation 3.32. Here we use a summation in the second term to allow for any size excitatory region.

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)\sum_j^m [I + J_j]z_j - (D + x_i)\sum_k^n [I + J_k]z_k \tag{3.32}$$

In this equation, the excitatory region is defined by the index j and has a size m. The inhibitory region uses the index k and has a size n (in general $m < n$).

The next step is to add the connection weights or kernels to our model. The mathematical representation of the model becomes;

$$\frac{dx_i}{dt} = -Ax_i + (B - x_i)\sum_j^m F_j^{(+)}[I + J_j]z_j - (D + x_i)\sum_k^n F_k^{(-)}[I + J_k]z_k \tag{3.33}$$

Here the functions $F_j^{(+)}$ and $F_k^{(-)}$ represent the discrete excitatory and inhibitory kernels, respectively. The spacial kernels are gaussian functions and can be described by the gaussian equation. Equations 3.34 and 3.35 give the continuous and discrete form of a gaussian curve. These equations are given in a form that ensures the area under the curve (or the sum of the discrete values) will equal unity (normalized). This is important if we are concerned with preserving the power of the

input signal.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{3.34}$$

$$f(idx) = \frac{1}{\sigma dx\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{dx-\mu}{\sigma dx})^2} \tag{3.35}$$

The two-dimensional form of a gaussian surface involves the product of two, one-dimensional gaussian curves. Derivation of the two-dimensional, continuous gaussian is given below.

$$f(x,y) = G(x,\mu_x,\sigma_x)G(y,\mu_y,\sigma_y) \tag{3.36}$$

$$= \frac{1}{\sigma_x\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{x-\mu_x}{\sigma_x})^2} \frac{1}{\sigma_y\sqrt{2\pi}} \exp^{-\frac{1}{2}(\frac{y-\mu_y}{\sigma_y})^2} \tag{3.37}$$

$$= \frac{1}{\sigma_x\sigma_y 2\pi} \exp^{-\frac{1}{2}(\frac{x-\mu_x}{\sigma_x})^2 - \frac{1}{2}(\frac{y-\mu_y}{\sigma_y})^2} \tag{3.38}$$

$$= \frac{1}{\sigma_x\sigma_y 2\pi} \exp^{-\frac{1}{2}\left[\frac{(x-\mu_x)^2}{\sigma_x^2} + \frac{(y-\mu_y)^2}{\sigma_y^2}\right]} \tag{3.39}$$

The discrete form of Equation 3.39 can be found by replacing the σ_x and σ_y with the discrete indices $\sigma_x dx$ and $\sigma_y dy$. To determine a value for each σ we use the fact that 99.9 percent of the gaussian falls within 3σ of μ [Kreyszig, 1988:1213]. Therefore, since we know the number of nodes we would like to span the curve, solving for σ becomes nearly trivial.

$$\text{Nodes} = 6\sigma \tag{3.40}$$

$$\sigma = \frac{\text{Nodes}}{6} \tag{3.41}$$

The two-dimensional gaussian given in Equation 3.39 does not simplify to a one-dimensional gaussian if the number of nodes in the y direction is set to zero because of the σ_y in the denominator

of the equation. This is a result of using a normalized gaussian curve. Figures 3.8 and 3.9 show one- and two-dimensional plots of the discrete spacial kernels.

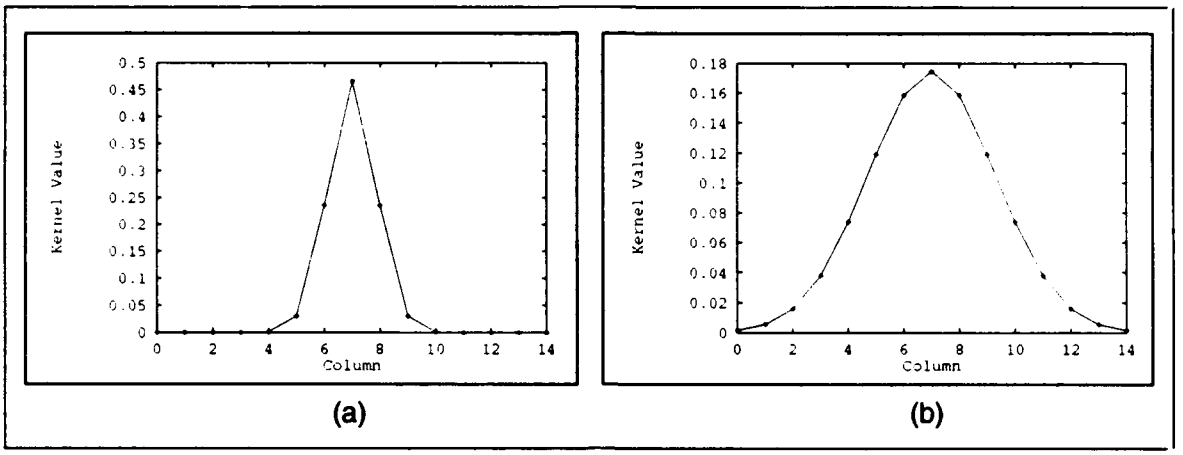

Figure 3.8. The spacial kernels defined in one-dimension. This figure shows a discrete excitatory (a) and inhibitory (b) kernel defined on a receptive field of 15 with an excitatory region of 5.

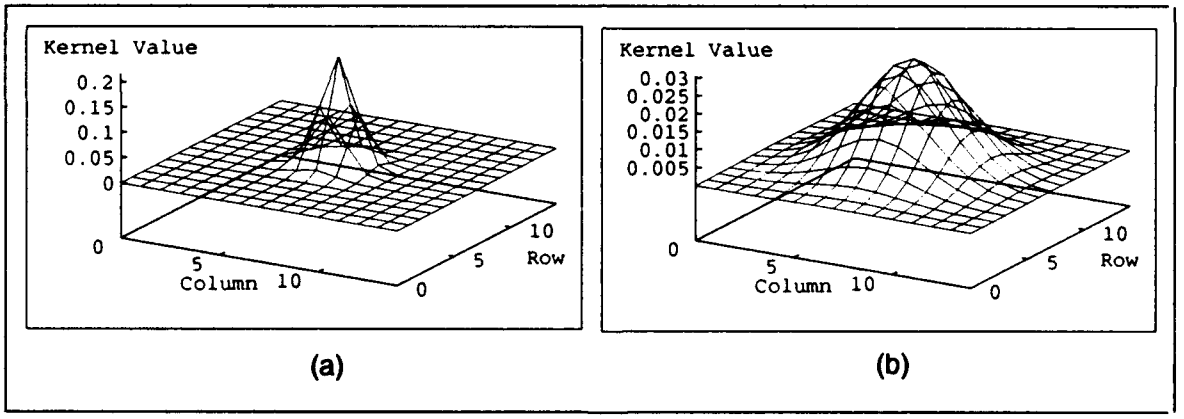

Figure 3.9. The spacial kernels defined in two-dimensions. This figure shows a discrete excitatory (a) and inhibitory (b) kernel defined on a receptive field of 15 with an excitatory region of 5.

Equation 3.33 describes the receptive field of a single neuron. This field is illustrated in one dimension in Figure 3.10. Here, there are three cell activity nodes (soma) and nine neurotransmitter nodes (synapse). The field size is three on either side of the soma or seven nodes total. The modeling software used for our studies sets the inhibitory region equal to the receptive field size. The excitatory region must be equal to or less than this value. Figure 3.11 is a diagram of the

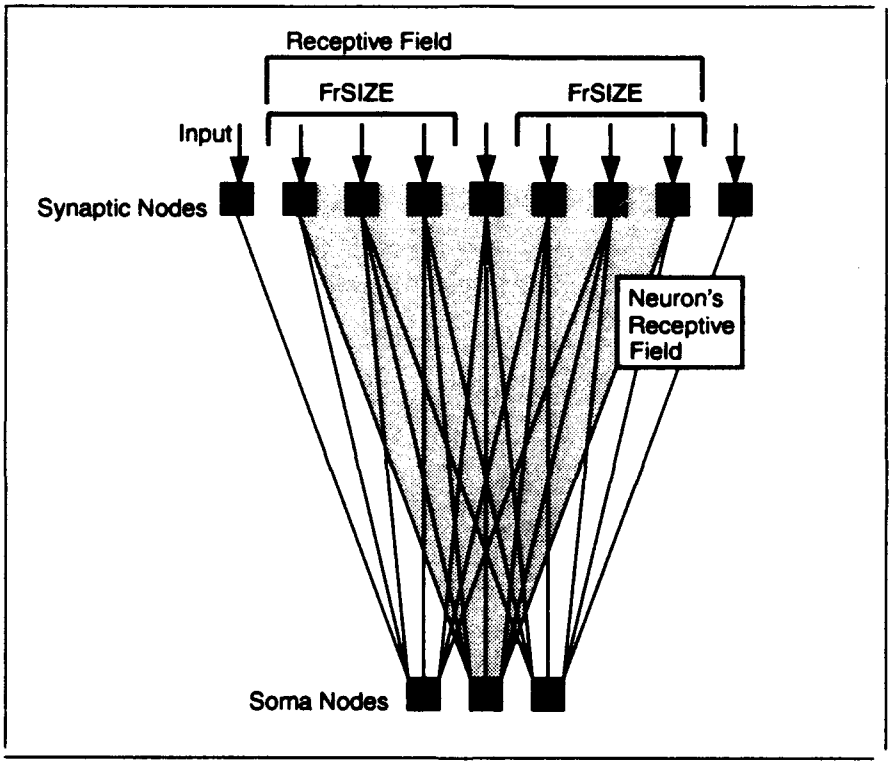

Figure 3.10. This figure shows how nine synaptic nodes are interconnected to three soma nodes to form the plexus. The receptive field size is seven synaptic nodes. The figure shows that the output array is six nodes less than the input. We reduced the input by twice the size of the receptive field's lateral extension to avoid dealing with edge effects.

one-dimensional receptive field showing the opposing regions. If signals from the two regions reach

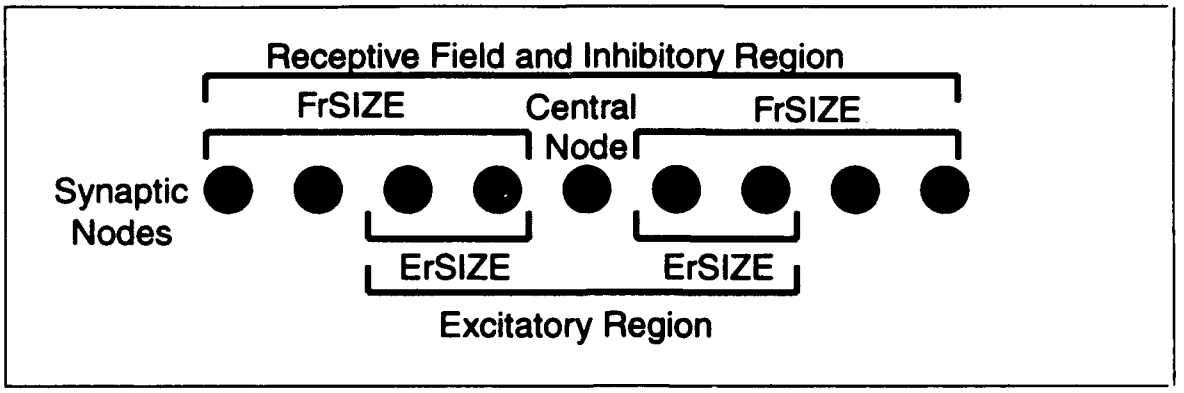

Figure 3.11. The Receptive Field of the Neuron. Here we show how the full receptive field is defined with four input nodes (FrSIZE) flanking the central node. The excitatory region is defined as two input nodes (FeSIZE) on either side of the central node.

the soma node simultaneously, then the model is synchronous. If the inhibitory region is delayed, the model is asynchronous. We can form two variants of either model by including or omitting the synaptic nodes. If they are omitted, the models are configured as Öğmen and Gagné presented in their paper. We add the synaptic nodes to form a second configuration. The former is termed the **O&G** model while the latter is the **adaptive** model. Both can be synchronous or asynchronous. The differences in the four models are best explained using graphical methods.

3.3.2 Model Response. We validated the software model in a one-dimensional configuration. This section uses those results to illustrate the differences between each form of the model. We begin with the synchronous model.

3.3.2.1 Synchronous Model Response. The input to our synchronous model is shown in Figure 3.12. Figure 3.13 illustrates the response of the O&G synchronous model to the moving RECT input. Figure 3.13 is annotated to identify certain common features of our model's response. The first is the *background plane*. This is the ambient surface from which the response deviates. Edges appear as a spacial overshoot located next to a spacial undershoot. The term *spike* is given to the spacial overshoot and *trough* to the spacial undershoot. We make this distinction to aid in

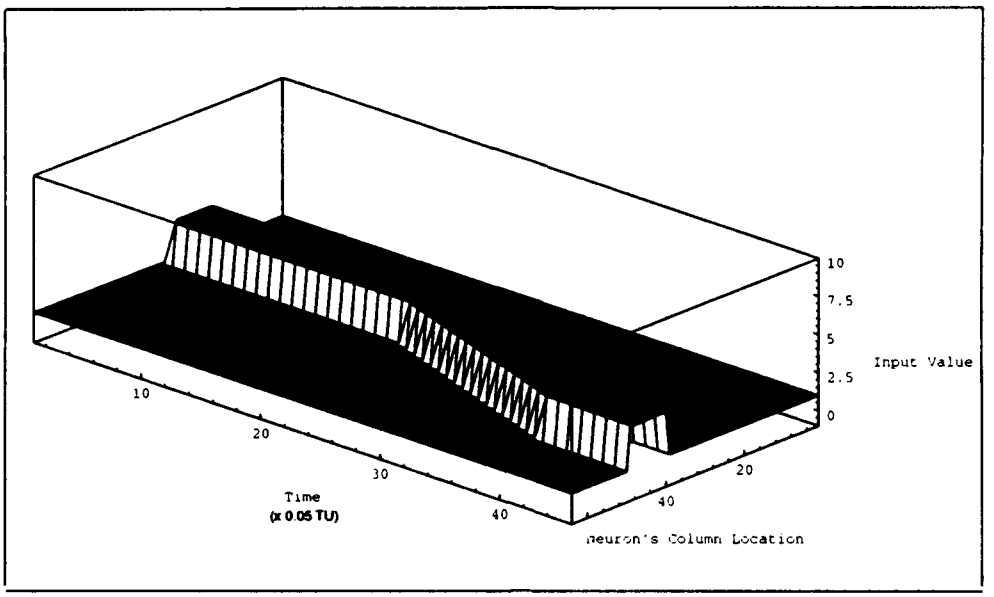

Figure 3.12. This figure shows how the rectangular input (RECT) is moved across the field-of-view of the model over time. Initially the model is forced into steady-state with a flat input pattern. Immediately after initialization, the RECT appears and remains stationary for 1.0 time units (TU). In the period from 1.0 to 2.0 TUs, the RECT is moved one node every 0.05 time units. In the period from 2.0 to 2.3 TUs, the RECT remains stationary. The speed of this RECT is 20 N/TU.

our discussion of the salient features of our graphs. Therefore, the terms undershoot and overshoot imply a spike or trough in time only. The area between any two spikes which is elevated from the background plane is referred to as the *plateau*. Three more terms will be introduced as we discuss the adaptive model response.

Figure 3.14 shows the response of the adaptive synchronous model. This figure shows that the adaptive response to a moving RECT is slightly different than the O&G synchronous model. Here we see the model adapting (has a reduced output value) to the stationary input until motion begins. When the RECT begins to move, the model responds more vigorously. When the RECT is in motion, the adaptive model has a markedly different form than that of the O&G model. Here, there is a forward, positive wave created by the overshoot of the neurons and a rear, negative wave created by the undershoot of the neurons. The region between the waves is slanted and is called the *mesa*. Notice the raised spike at the location where the RECT begins to move. This is the result

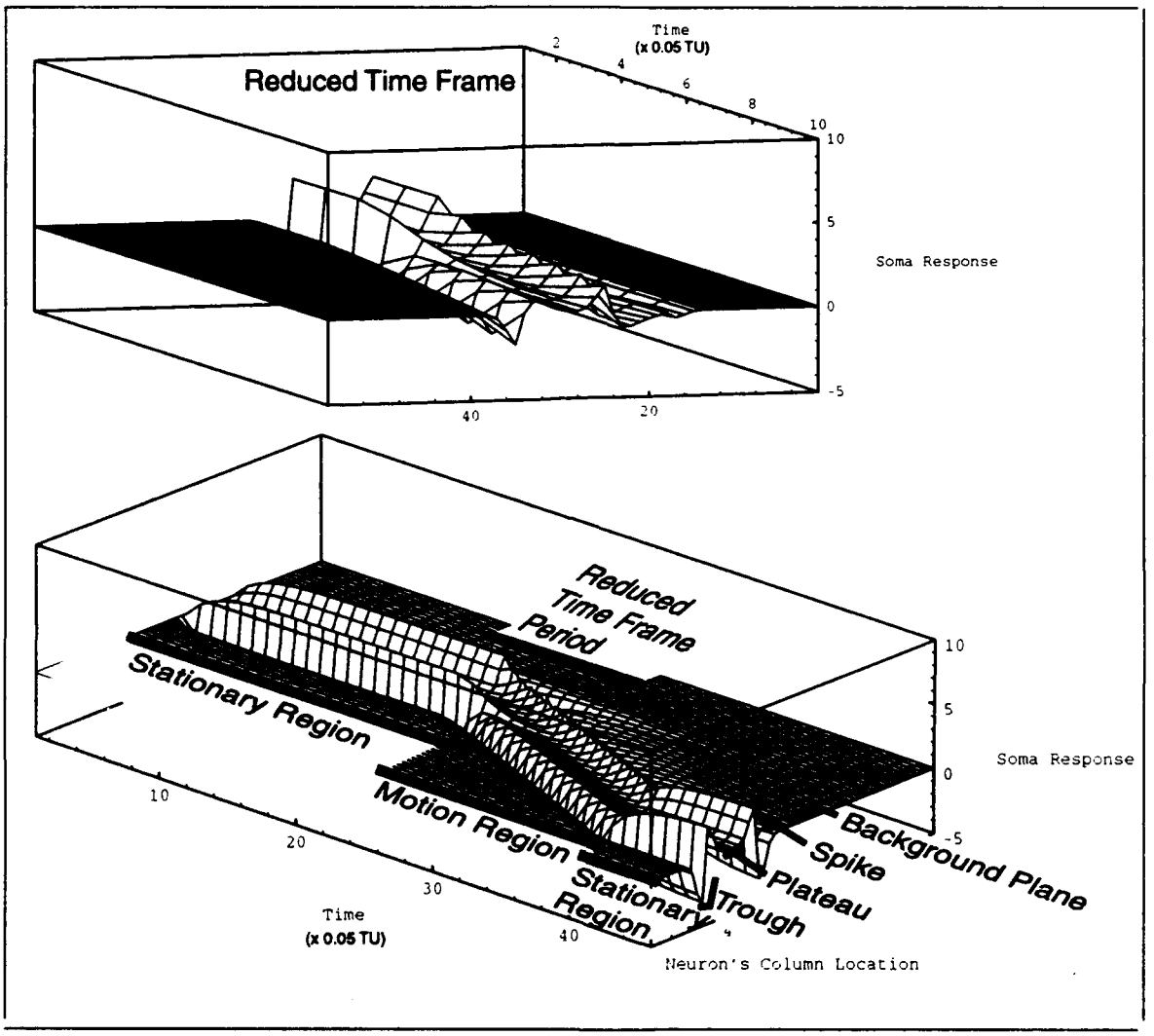

Figure 3.13. This figure shows how the Öğmen and Gagné synchronous model responds to the rectangular input (RECT). The lower graphic is the full model output and the upper graphic is a short period extracted to show the form of the wave at some time during the motion period. As we noted in Chapter II, the model responds more vigorously to stationary inputs and is less responsive to moving inputs. Some features of the plot such as the *background plane*, the *trough*, the *plateau*, and the *spike* are annotated here.

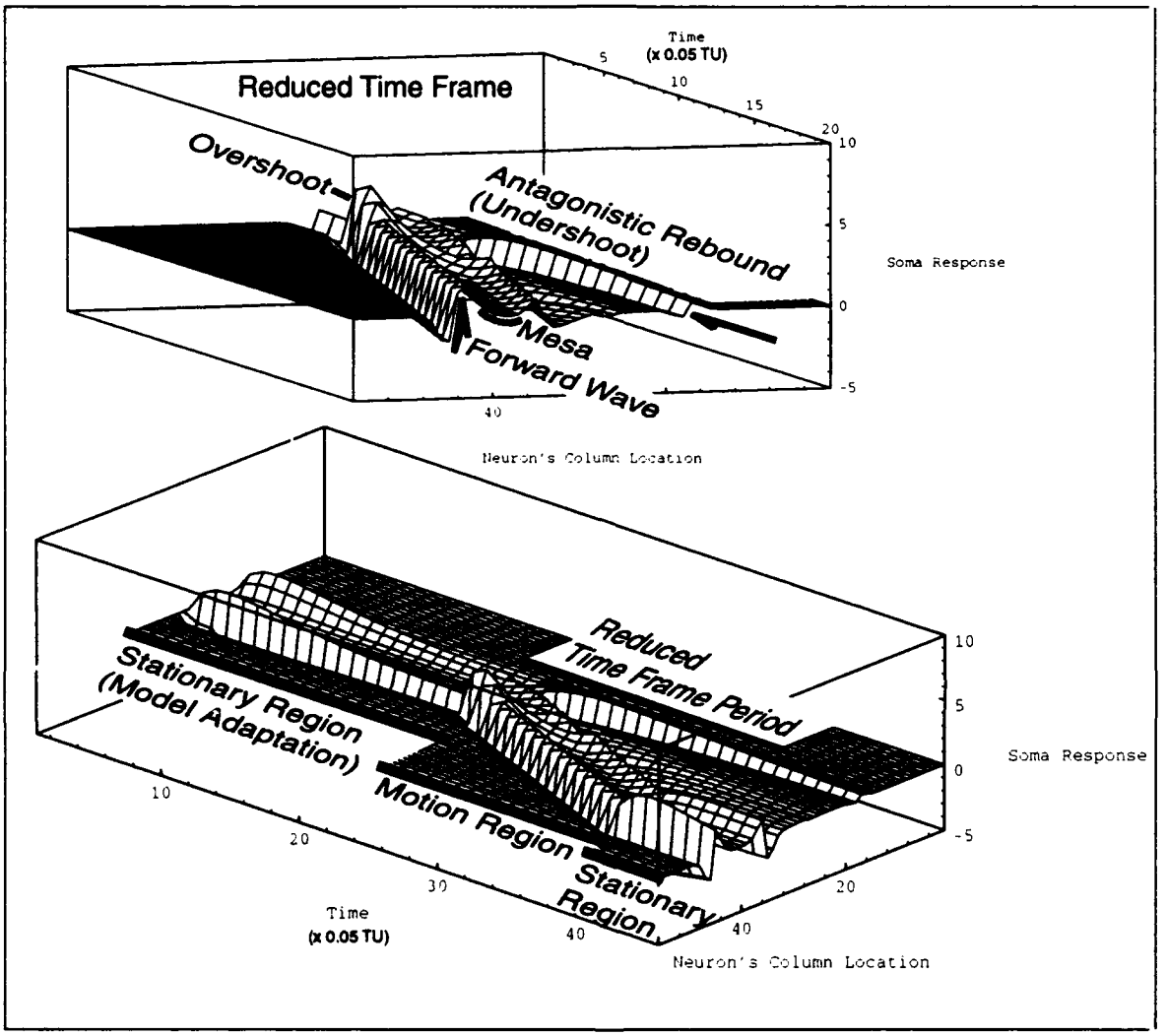

Figure 3.14. This figure shows how the adaptive synchronous model responds to the rectangular input (RECT). The lower graphic is the full model output and the upper graphic is a short period extracted to show the form of the wave at some time during the motion period. The figure shows the model adapting shortly after the introduction of the stationary RECT. When the RECT moves, the model response increases momentarily and begins to adapt once again. In this figure, the jagged edges of the motion *wave* are caused by our discrete model. We see some antagonistic rebound in the figure as the RECT enters the trough region of the stationary response. The *mesa* and the forward wave are identified here.

of the elevated stimuli entering the previously depressed trough of the stationary RECT. That is, the RECT has a greater effect on the region which adapted and is currently at a lower value than it does in the region where the background plane exists. Likewise, the RECT edge which moves into the plateau region shows little change but the plateau region drops slightly before the edge reaches it. This is due to the change in input value over the receptive field of the neurons in the plateau region. The region outside the trailing edge experiences antagonistic rebound as is evident by the ripples in the background plane. Once the RECT stops, the model begins to adapt once again. These features are exaggerated in the asynchronous forms of the models.

3.3.2.2 Asynchronous Model Response. The input to our asynchronous model is slightly different than the input used for the synchronous model. Figure 3.15 shows the input used. The difference is the amount of time the RECT is stationary. In this figure, the RECT remains stationary for only 0.05 TU initially while the RECT stops for 0.8 TU in the end. This change helps to show the final adaptive behavior of these models.

Figure 3.16 illustrates the response of the O&G asynchronous model to the moving RECT of Figure 3.15. We can see that this asynchronous version of the O&G model has a greater response to the introduction of the RECT than the O&G synchronous model. After the initial transient of introducing the RECT, the model decays to an elevated plateau sided by spikes and troughs. When the RECT begins to move, the model forms a set of waves and mesa at an elevated level which did not occur in the previous O&G model. In the final stationary period, the model again decays to the plateau, spike, and trough form found in the last TUs of Figure 3.13. The final two figures of this section illustrate the features of the adaptive asynchronous model response.

Figures 3.17 shows the response of the adaptive asynchronous model. This model can be compared to the O&G asynchronous model. Both models have an initial overshoot at the introduction to the RECT but the adaptive model decays significantly more and the plateau and spikes almost blend into the background plane. During the RECT's motion, the forward wave and mesa

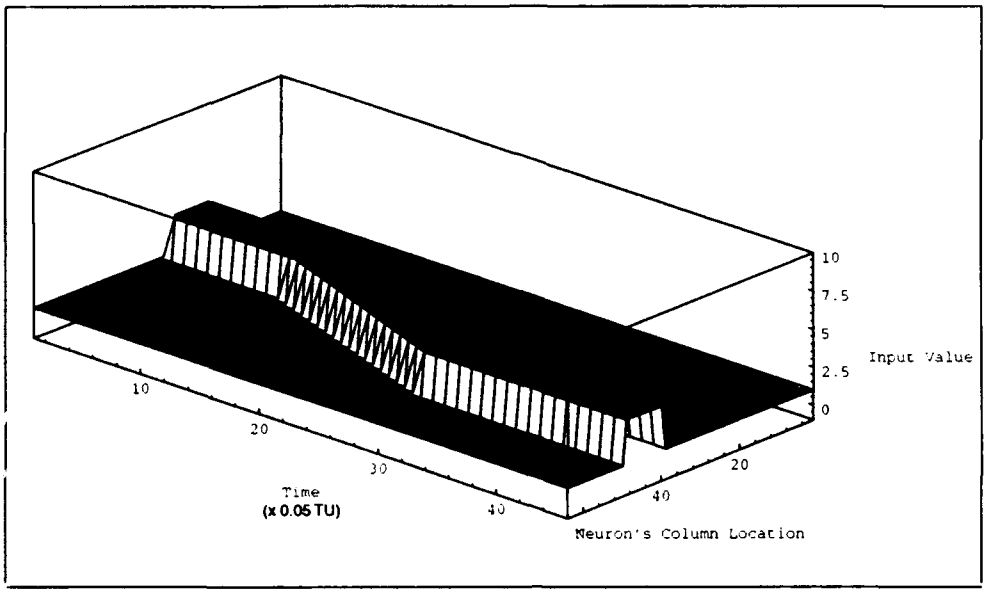

Figure 3.15. This figure shows how the rectangular input (RECT) is moved across the field-of-view of the model over time. Initially the model is forced into steady-state with a flat input pattern. Immediately after initialization, the RECT appears and remains stationary for 0.5 time units (TU). In the period from 0.5 to 1.5 TUs, the RECT is moved one node every 0.05 time units. In the period from 1.5 to 2.3 TUs, the RECT remains stationary. The speed of this RECT is 20 N/TU.

are larger than that of the O&G model . Finally, the model adapts closer to the background plane than does the O&G model in the final TUs of the simulation.

In all, we see that the adaptive asynchronous model performs better than the O&G asynchronous model in relation to identifying transients and motion. It also decays closer to the ambient (background plane) than the O&G model does. In the next section, we look at some methods of post processing the model response to highlight transient behavior.

3.4 Post Processing Model Output

In a single dimension we can clearly see where motion is occurring. However, when we extend the model to two dimensions and the output has to be represented as image sequences, the features of the response are easily lost among the 256 gray-levels used to represent the cell-activity level. The spikes and troughs are still apparent but the more subtle features like mesas, plateaus, and

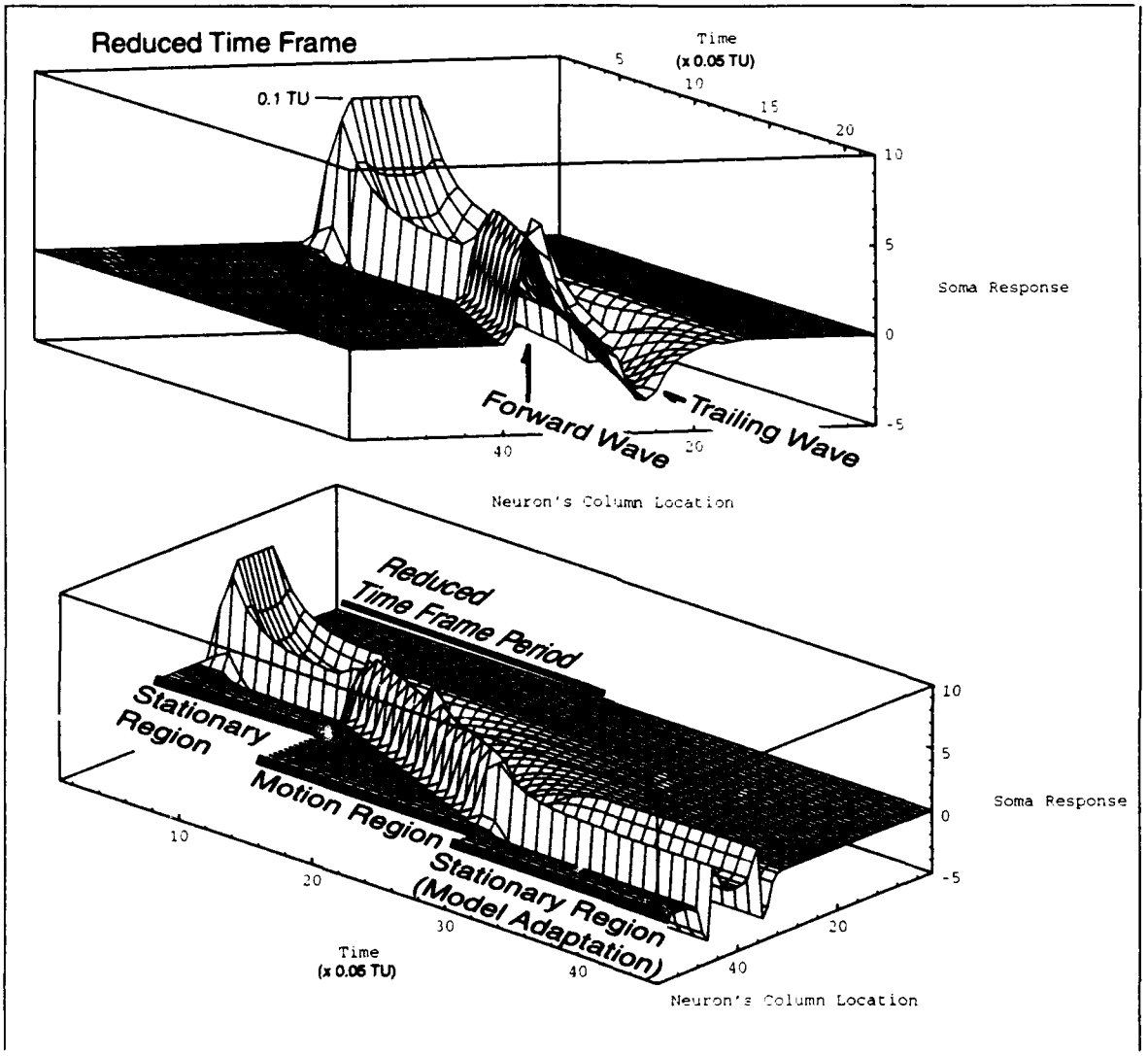

Figure 3.16. This figure shows how the Öğmen and Gagné asynchronous model responds to the rectangular input (RECT). This figure is scaled identically to Figure 3.13. The lower graphic is the full model output and the upper graphic is a short period extracted to show the form of the wave at some time during the motion period. Here we see the inherent adaptive behavior of the asynchronous model. The model overshoots and adapts to the introduction of the RECT. It tracks the RECT's motion with a positive wave and a negative wave, and then adapts to the final stationary period.

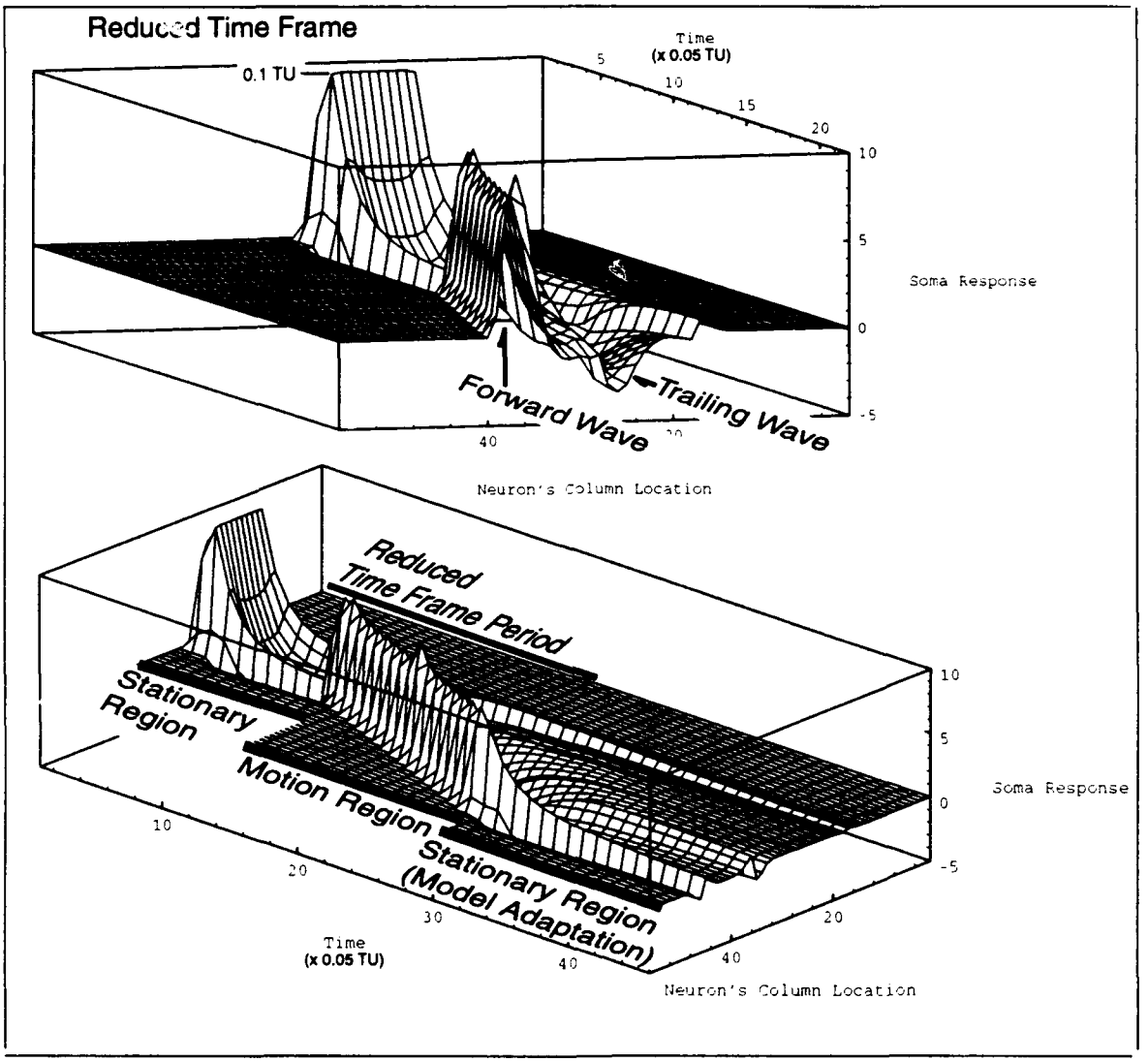

Figure 3.17. This figure shows how the adaptive asynchronous model responds to the rectangular input (RECT). The lower graphic is the full model output and the upper graphic is a short period extracted to show the form of the wave at some time during the motion period. These figures show the model adapting shortly after the introduction of the stationary RECT. When the RECT moves, the model response increases momentarily and begins to adapt once again. In this figure, the jagged edges of the motion wave are caused by our discrete model. The mesa and the forward and trailing waves are illustrated here as well. In the final TUs, the model adapts to the stationary RECT.

rippling in the background plane are difficult to identify. In the next chapter we will be solely interested in the model's ability to identify motion. Therefore, we developed two techniques of post processing the output data to focus onto transient behavior on the model input.

The two post processing methods are moving average subtraction and thresholding. We use four types of moving averages. They are a causal and noncausal level-window average and a causal and noncausal gaussian-window average. Figure 3.18 illustrates the four types of moving averages. The figure shows nine images spanning a "window" in time centered on the image currently being

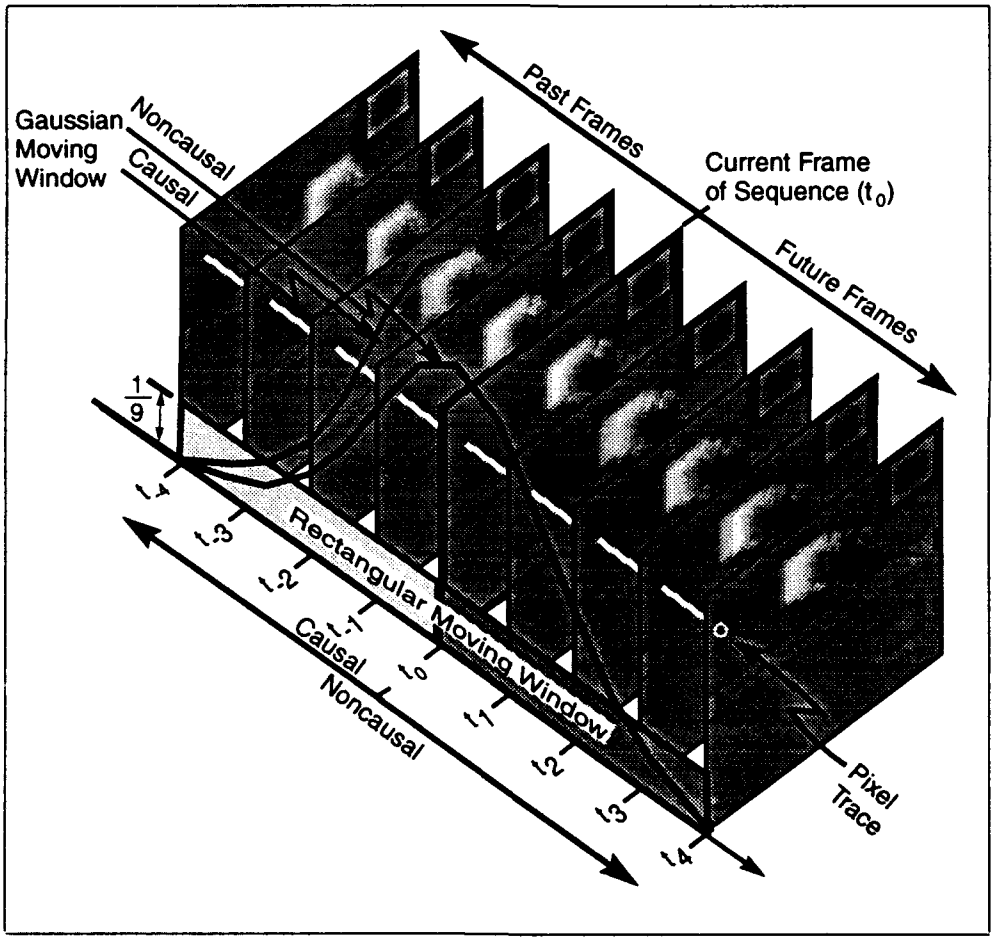

Figure 3.18. This figure illustrates the moving window averages used in our research. The figure contains nine frames of output. The center frame is the image currently being processed. In the causal moving average, the past four frames are only used. Either a straight average is determine for each pixel trace or a gaussian weighted average is taken. In the noncausal case, the windows extend into the future four frames. This average can be rectangular (level-windowed) or gaussian-windowed.

processed. A pixel trace is shown as a line connecting a single pixel through time. The moving average process calculates an average for each pixel trace and subtracts it from the current pixel value. In the causal moving average, the averaging window extends only into the past. No future data is needed in this process and the window can be rectangular (level-windowed) or gaussian in shape. The noncausal moving average uses either window type but adds as many future images as past images. This scheme requires the model to delay its output until it can calculate the current frame value. To show how these techniques effect the model response graph, we have the output from a simulation in which a small, stationary RECT is crossed by a larger, moving RECT (Figure 3.19). The response is given in Figure 3.20. This figure has a ridge running from the time

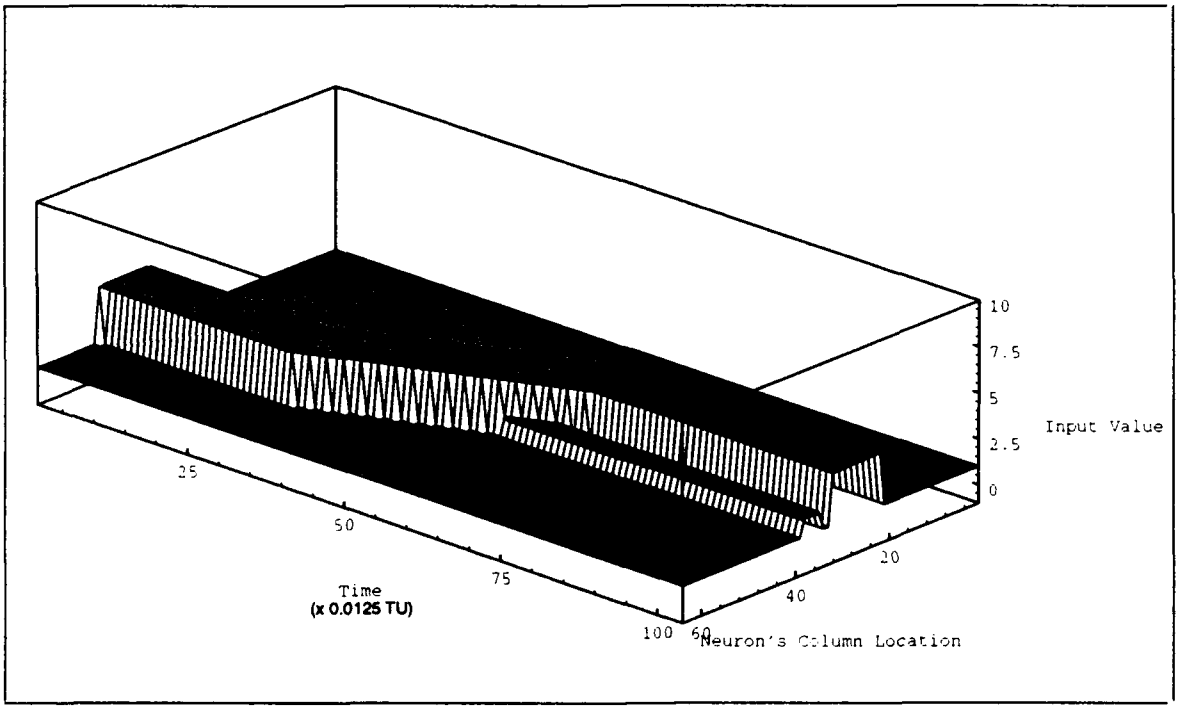

Figure 3.19. The input depicted here is of a larger RECT crossing over a stationary smaller RECT.

zero until the motion begins. This and the dip in the forward wave when the RECTs cross disturb us. We applied the causal and noncausal, level-window, moving averages to this response.

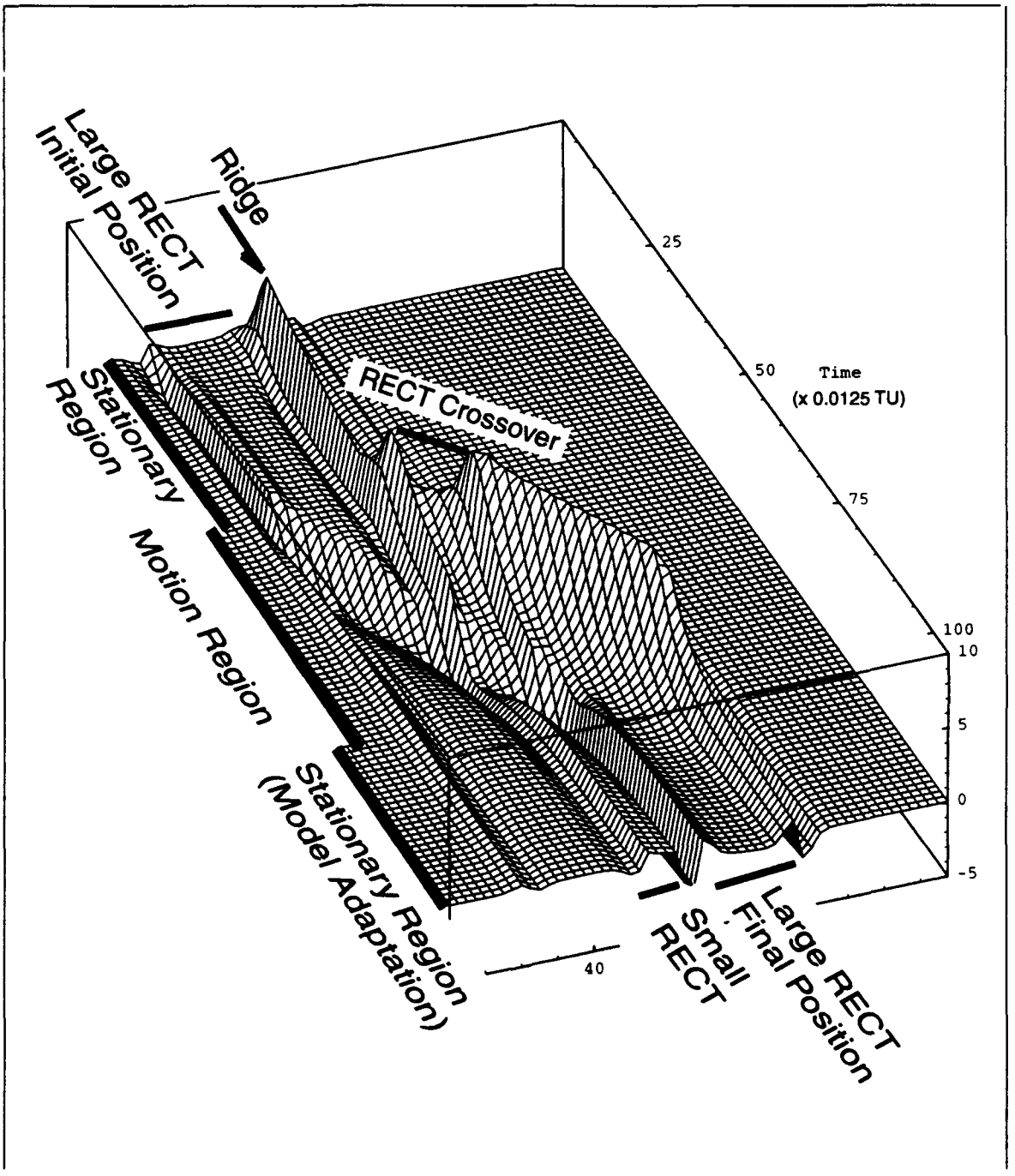

Figure 3.20. Response of a one-dimensional model to an input sequence containing a large moving RECT crossing a smaller, stationary RECT. This is the adaptive model's output. Here we see that the location where the large RECT crosses the smaller RECT forms a dip in the forward moving wave. We use this result to study the effect of moving average and threshold post processing techniques.

When we used the causal, level-window, moving average on the output we found the plateaus, mesas, and rippling was severely diminished as expected. Figure 3.21 shows the results of this process. The height of the response overall was also decreased but the initial transient and the forward and trailing wave were preserved. In fact, these features were enhanced in relation to the surrounding features in the final graphs. The ridge is no longer evident but the background plane swells following the trailing wave trough. This will have an adverse effect when we attempt to threshold later. The noncausal, level-window, moving average has a surprising response.

Figure 3.22 illustrates the result of post processing the data through a noncausal moving average. The most intriguing feature in this result is the inversion of the trailing wave. Here, the trough that used to travel along the trailing edge of the RECT is turned into a spike or ridge. The result of this inversion is a definite plateau between the forward and trailing waves. Each wave has a spike and trough identifying the edge of the moving RECT. Except for the initial transient, this process identifies only motion. The background plane remains nearly planner outside of the motion region. To show the importance of a flat background plane, we apply a threshold operation to the model response.

The threshold we apply to our models is of two types. The first is a lower-plane truncation. That is, we set any value lower than the threshold value equal to the threshold value. We truncated those lower values. A second type of threshold is a windowing threshold. Like the time window, we apply a response window which truncates anything greater than the threshold (T) and anything lower than the negative of the threshold ($-T$). Only those values within the bounded interval $[T, -T]$ remain and all others are truncated. The result of the lower-plane truncation is given in Figures 3.23 and 3.24 for the causal and noncausal post-processing graphs respectively. The causal graph shows a region of the background plane still evident after the process. The noncausal graph clearly shows the forward and trailing waves as well as a trough preceeding the wave. We also see the location of the crossover and the initial transient. Our use of thresholding is not meant to

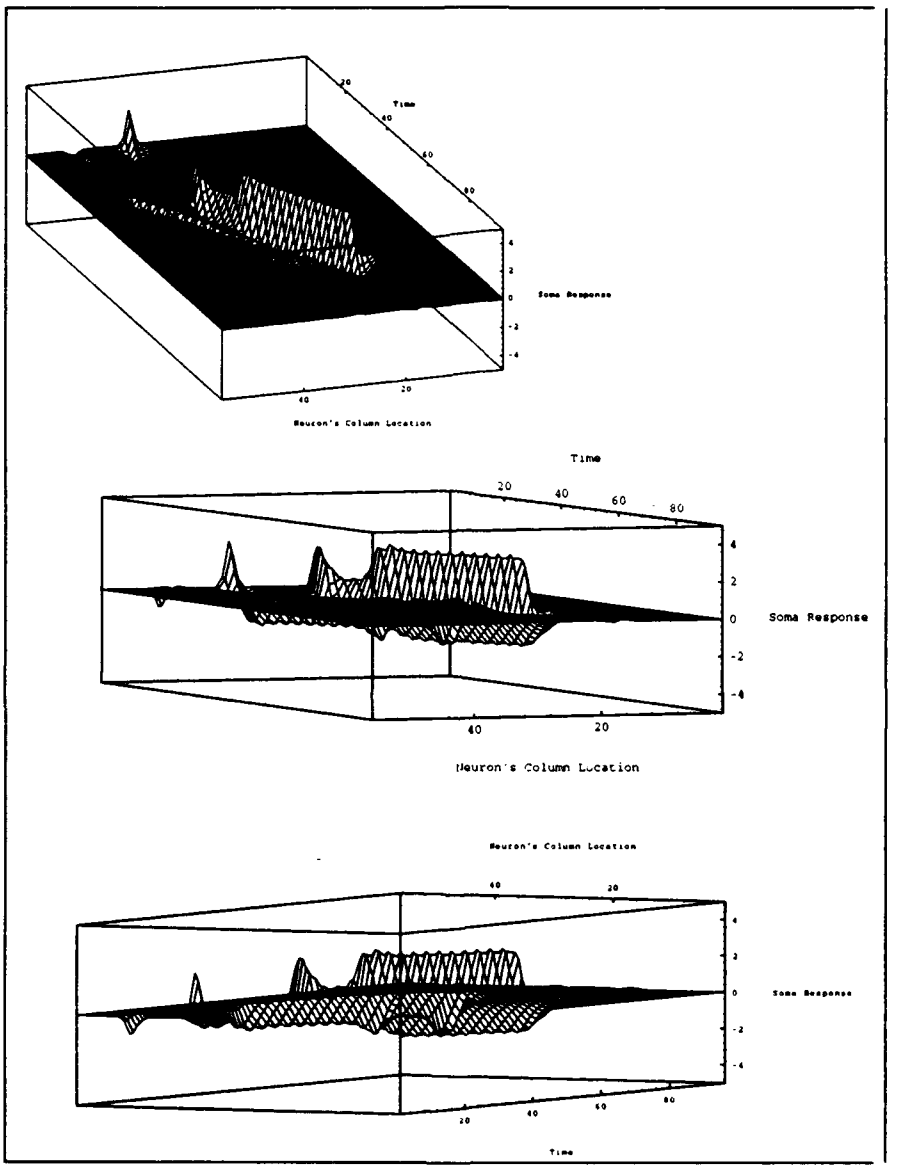

Figure 3.21. Subtracting the Moving Average from the Current Frame (Causal). The result of subtracting the causal moving average from the current output frame of the model's response. This figure shows the graph from three angles. The upper plot is viewed from well above the plane of the time-column origin. The middle plot is viewed from slightly above the plane of the origin while the lower plot views the plot from below.

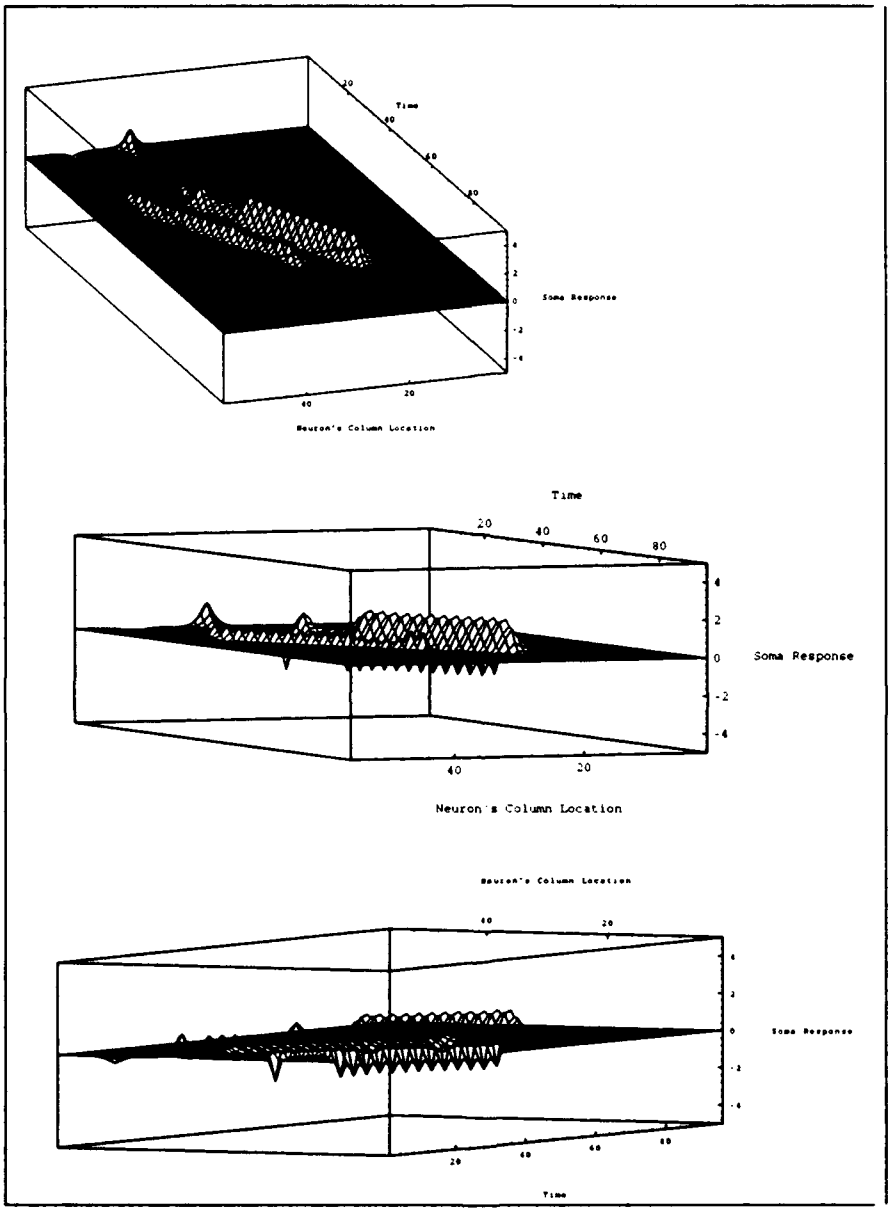

Figure 3.22. Subtracting the Moving Average from the Current Frame (Noncausal). The result of subtracting the noncausal moving average from the current output frame of the model's response. This figure shows the graph from three angles. The upper plot is viewed from well above the plane of the time-column origin. The middle plot is viewed from slightly above the plane of the origin while the lower plot views the plot from below.

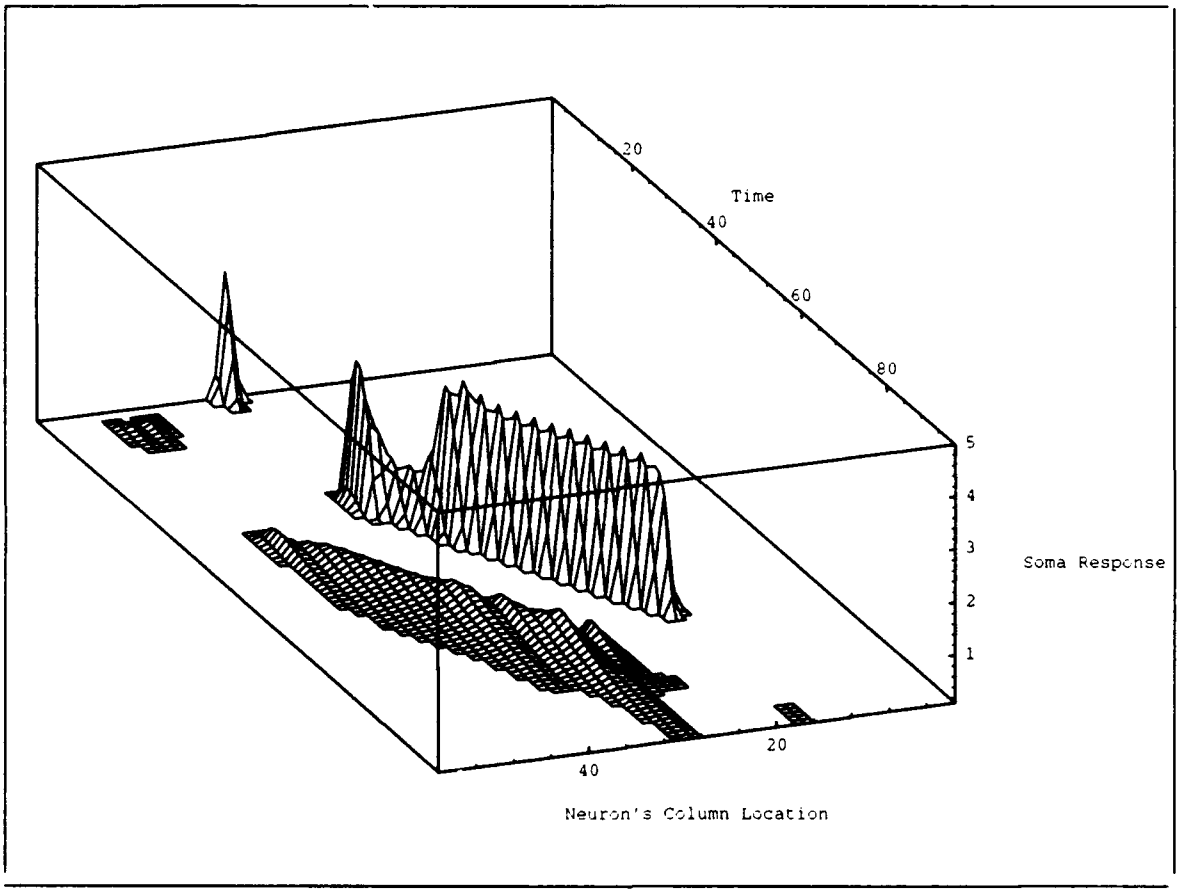

Figure 3.23. The result of thresholding the causal moving average postprocess output of the model response at a value of 0.1.

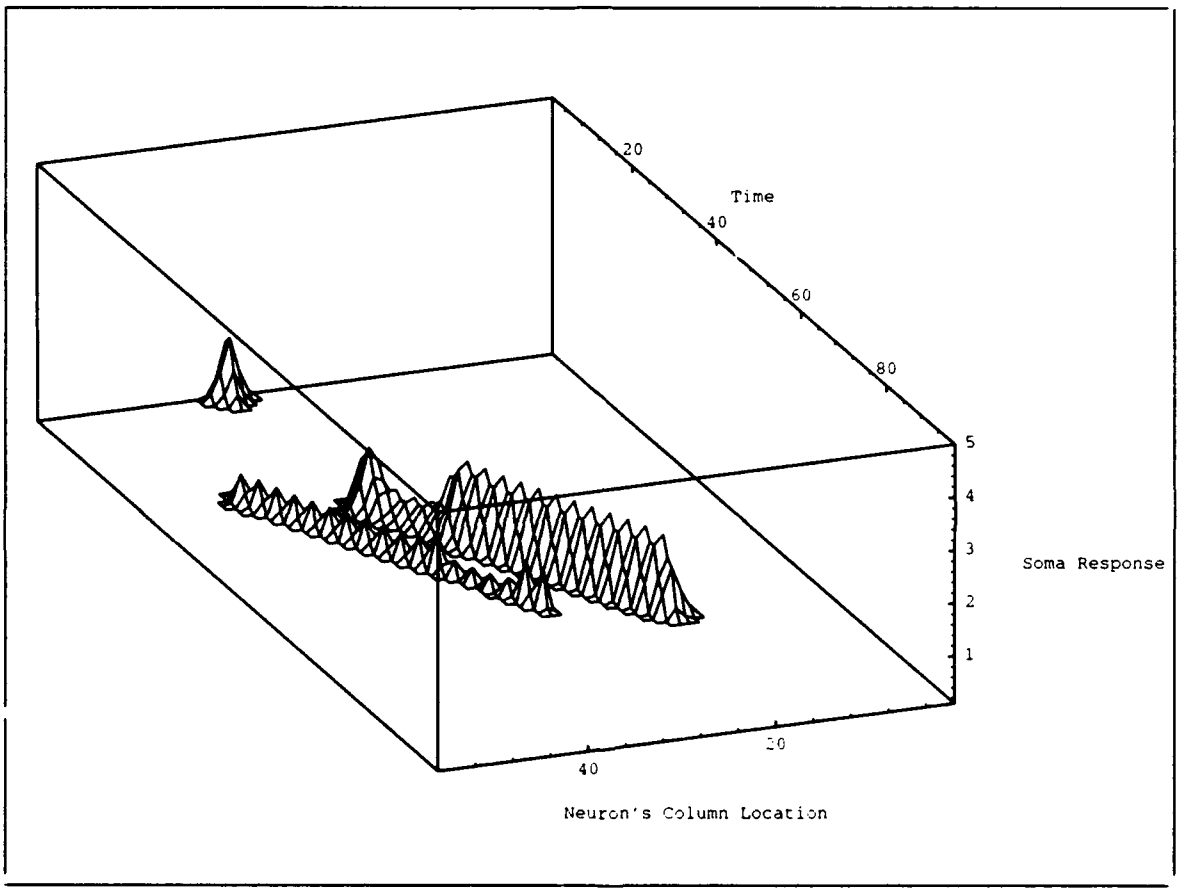

Figure 3.24. The result of thresholding the noncausal moving average postprocess output of the model response at a value of 0.1.

be indicative of the necessary processing for our model response. We include it to show that the output can be further processed to gain more information about the image sequence. The causal moving average was less effective than the noncausal filter. In the latter, the output clearly showed both the leading and trailing edges of the RECT while providing a trough in front of the RECT. This trough indicates the future position of the RECT. We get this result because the noncausal filter incorporates future images into the current output. The problem with thresholding is that the value chosen is a function of the imagery under investigation. We could choose any value to threshold with and get varying results. We might choose a threshold that eliminates the dip and the initial transient spike. The difficulty here is to choose a threshold that is appropriate to the input, in other words, a dynamic threshold. We do not impliment such a threshold in our work. In fact, we do very little with thresholding from here on out. In the next chapter we apply a gaussian-window, moving average and windowed thresholding to our two-dimensional model response.

3.5 Summary

In this chapter we explained the two basic models used to build our model. In the neurotransmitter model, a first-order, linear, time-variant, differential equation characterizes the synaptic junctions between the transmitting and receiving neuron. This equation provides an adaptive response to the changing input pattern. Its adaptation is slow and therefore the equation forms a long-term memory for the cell. The form of the model's response is a slow decay from the model's steady-state, or resting potential to an adapted potential. The neurotransmitter model is used in our model to form each synapse which is represented as a node in the model architecture. The synaptic nodes feed the second-order nodes which use the cell-activity model to determine their response to stimuli.

The cell-activity model, is also a first-order, time-variant, differential equation. The solution to this equation was made complicated by the summation terms which form the receptive field. Since

the dynamics of the cell activity model turn out to be much faster than that of the neurotransmitter model, we can determine the steady-state value of the response. In a simplified architecture, the gated dipole, the model has two neurons with receptive fields extending to the immediate neighbor. We showed that the model responds with an overshoot at the introduction of an input to one of the neurons. The models eventually decays to a plateau. When the input is removed, the neighboring neuron rebounds and the model settles into it original state. The cell activity model was termed the soma for purposes of model development. Having shown this behavior, we turned to the two-dimensional realm.

Using the two basic model types, we showed how the gated dipole was extended to produce an excitatory and inhibitory region over the input synaptic nodes. The signal from each node in either region was weighted by a two-dimensional gaussian. Taken as a unit, each soma and the corresponding synaptic nodes of its receptive field was termed the neuron. We developed four models with these neurons. Two synchronous and two asynchronous. The results of preliminary simulations were given to explain the model's behavior to moving objects projected onto the input plane.

In the last section, we looked at methods of post processing the model response to enhance transitory inputs over stationary ones. The two methods of post processing were to use a moving average (causal and noncausal) and to threshold the output. In the next chapter we will examine the results of our model to two types of input sequences. The first sequence represents a pristine image. The second is real FLIR imagery of a tank moving around the field-of-view. We describe each input in full and then give the results of our model's response. After explaining the output, we show how the post-processing techniques affect its appearance.

IV. Model Performance

In Chapter III we illustrated how our model differs from that proposed by Öğmen and Gagné. We implimented the neurotransmitter model at the input node allowing our model to adapt to its input. Therefore, we called this the adaptive model and compared our one-dimensional anaylsis to the model proposed by Öğmen and Gagné (O&G Model). Since the nonrecurrent, asynchronous forms of the models provide superior results over their synchronous forms in one dimension, we continue to investigate the response of these models in two spacial dimensions. Neither the adaptive or O&G model has been studied in two-dimensional before.

In this chapter we apply our models in their two-dimensional form to two image sequences representing motion. The pristine imagery is computer generated and contains no noise. In the first half of the chapter we examine the response of our models to this imagery and discuss the effectiveness of the model and post processing techniques. The second half of the chapter deals with simulations run on real Forward Looking Infrared Radar image sequences. This last set of inputs gives us an idea of how the model will respond to real inputs. This will conclude our results leading us into Chapter V and our conclusions.

4.1 Simulations Using Pristine Imagery

In our first simulation, we use a set of images that were artificially created in a *virtual* environment. Therefore, all objects in the scene have well defined edges and motion, the scene is free of noise, and the field-of-view (FOV) remains stationary. This "pure" form of images is referred to as a ***pristine input sequence.*** Figure 4.1 depicts the motion apparent in the pristine input. This figure depicts the motion created by our pristine image sequence. It also shows two other elements of the scene - a vertical bar and an oscillating box.

The motion shown in Figure 4.1 by the numbered boxes, is a sweeping curve from the lower right corner of the image to the center of the top of the image. The box moves in an arc that

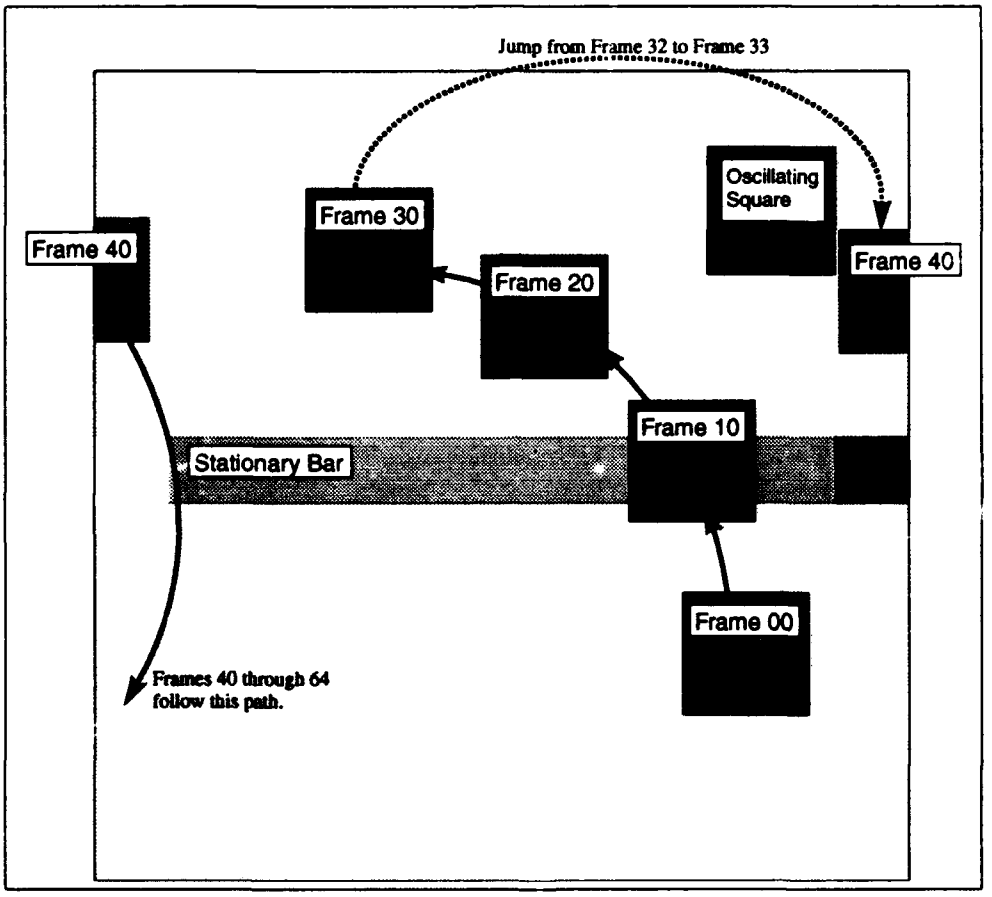

Figure 4.1. This figure shows a moving box crossing a vertical bar of half its intensity. The box moves at 2.8 degrees per frame and the simulation is made-up of 64 frames. A box in the upper right corner remains stationary but oscillates over time in brightness.

Table 4.1. Intensity values used for the object in our pristine images.

Object	Intensity	Size
Moving Box	4.0	10 x 10 pixels
Oscillating Box	0.0 - 16.0	10 x 10 pixels
Bar (Center)	2.0	5 x 52 pixels
Bar (Cold End)	0.0	5 x 6 pixels
Bar (Hot End)	16.0	5 x 6 pixels
Background Plane	1.0	64 x 64 pixels

spans 90 degrees in 32 frames or 2.8 degrees per frame. After 32 frames, the box jumps to a new location in the upper right corner of the image and sweeps downward out of the FOV and into the opposite side. The motion of the box is smooth. Unlike the motion of the RECTs in Chapter III, we do not jump from one location to the next as the images are applied to the model input. In this simulation, we allow the edge of the box to cover only a portion of the input node. Thus, the edge of the box may look less intense to the node as it moves into the nodes FOV. Therefore, the introduction of an edge to the input node is gradual as is the removal of the trailing edge and the motion becomes smoother. We will see later that the edges of the input images are slightly blurred as a result of smoothing the motion of the box.

The moving box crosses a less intense vertical bar on both halves of its journey. The bar has a hot (white) and a cold (black) end to identify maximum and minimum image values. The last element of the image is an oscillating box in the upper right corner. This box changes intensity throughout the simulation. Over the 64 frame sequence, the box increases intensity over the first 32 frames and then decreases to its initial value in the last 32 frames. This is a linear ramp versus time. The values given to each object in the image are listed in Table 4.1.

We built this sequence to see how our models response to motion (primarily), intensity changes, minimum and maximum regions of the scene, and motion over various background intensities. The results of processing these images through both the O&G and adaptive model are addressed next. Although the simulation ran for 64 frames of input, we will limit the following

Table 4.2. Parameters used to configure the simulation software for the pristine input sequence.

Synapse Parameters		
α	0.5	Natural Decay Term
β	5.0	Target Value
SYNTOL	0.0005	RungeKutta Tolerance
Soma Parameters		
A	5.0	Natural Decay Term
B	45.0	Upper Bound
D	45.0	Lower Bound
SOMATOL	0.0005	RungeKutta Tolerance
TIME_DELAY	0.05	Inhibitory Delay
Neuron Size		
Receptive Field	9 Nodes	Vertical Direction
	9 Nodes	Horizontal Direction
Excitatory Region	3 Nodes	Vertical Direction
	3 Nodes	Horizontal Direction

discussions to the first 40 frames for two reasons. The first is to limit the amount of figures required to describe the model's response. The second is that the last 24 frames offer no new information from the first 40.

4.1.1 Model Parameters. The simulation results given in this section are from the application of both the O&G and adaptive models. Table 4.2 identifies the values used to configure the two models.

4.1.1.1 Predicted Results. Before we show the results of our simulations on the pristine imagery, we might stop and ask, "what do we expect to see as the model response." We expect several things to occur. First, and foremost, we expect the models to enhance the moving box and de- emphasize the bar (including the bar's ends). Since the receptive field is smaller than either the moving box or the oscillating box, we would expect the neurons at their centers to respond at a lower level than those whose receptive fields cover the edges of the boxes. The moving box may not show this behavior as well as the oscillating box because its movement will counteract the adaptive behavior of the models. We would expect the O&G model to have a darker center for the

moving box than the adaptive model because it is able to respond quicker. A neuron in the O&G model can reach its steady-state value in a shorter time and may be able to do so from the time the leading edge reaches its position and the time the trailing edge does. The oscillating box, however, changes intensity slowly and so both models may show a darker center.

Also associated with the speed of the neurons in these models is the time for the neurons to rebound from an adapted state to the resting potential. When the moving box leaves a neurons receptive field, the O&G model should return to normal quicker than the adaptive model. Thus, we should a dark region behind the moving box for a longer time with the adaptive model.

Returning to the bar, we can not expect the models to remove it from the output altogether. We saw in the linear data given earlier, that the models respond to stationary RECTs with a trough and spike feature straddling the RECTs edge. Therefore, we expect the sides of the bar to be surrounded by a trough on the outside of its edge and a spike on the inside. This same phenomena should occur at the boundary between the bar and its hot and cold ends. Since the linear simulations showed the adaptive model response to stationary inputs to be nearly the same value as the background plane than the O&G's response, we might expect the bar in the adaptive model to be less evident than in the O&G model. One last prediction before we look at the actual model output.

In Chapter III we used data from a simulation involving a large RECT passing over a less intense, small RECT in order to illustrate post processing techniques. In that figure (Figure 3.20) the response of the passing RECTs showed a dip in the output of the model due to a relative change in intensity of the moving RECT to its surrounding plane (which is now the lower intensity RECT). We expect this to occur in both the models. Now lets see how we did in our predictions.

4.1.1.2 Model Response. We begin our discussion of the simulation results by giving a synopsis of the model response as well as the results of passing the model output through a moving average filter. The two figures, can be used to compare the raw output data to the data adjusted by

the moving average algorithms. Figure 4.2 contains the data from our O&G model and Figure 4.3 is from our adaptive model simulations.

4.1.1.3 Analysis of Model Response. Lets begin by comparing the raw output data from either model (the second column of these figures). Even at the resolution provided here we can see the different response to the oscillating box. In the O&G model the box is well defined by a central, light region (plateau) surrounded by a dark band (trough). The box becomes more distinct as time goes on. Even after frame 32 when the box intensity begins to decrease, the O&G model picks it out. The adaptive model, however, adapts fairly well to the changing intensity all the way through frame 40. So, we were correct in predicting the O&G behavior but not as close with the adaptive model. In [Swanson, 1992] we showed that the response of the adaptive model to a stepped input resulted in a saturation of the model. When the model saturated, it could no longer respond to changes in the input. The O&G model, on the other hand, showed a trend toward saturation but *only very gradually.* Thus, the model was able to respond to changes in input intensity. We see this behavior in the frames given in Figure 4.2.

Our next prediction involved the low intensity bar. Neither model eliminated it completely, but the adaptive output comes very close. Looking at the O&G model, there is a dark outer band lining the bar's edges. We also see a pronounced dark region surrounding the hot end of the bar. The cold end has a very dim lighter region around its perimeter as well. The adaptive model shows nearly no sign of the hot end but does give indication of a lighted perimeter around the cold end. The absence of the hot end may also be attributed to the model saturating to the input. At the point were the moving box crosses the bar (frame 10), the bar shows up more clearly in the adaptive model as a diminished response of the boxes value. The O&G model, however, shows little of this behavior. In order to better investigate this feature, we look at magnified views of frame 10 and frame 15.

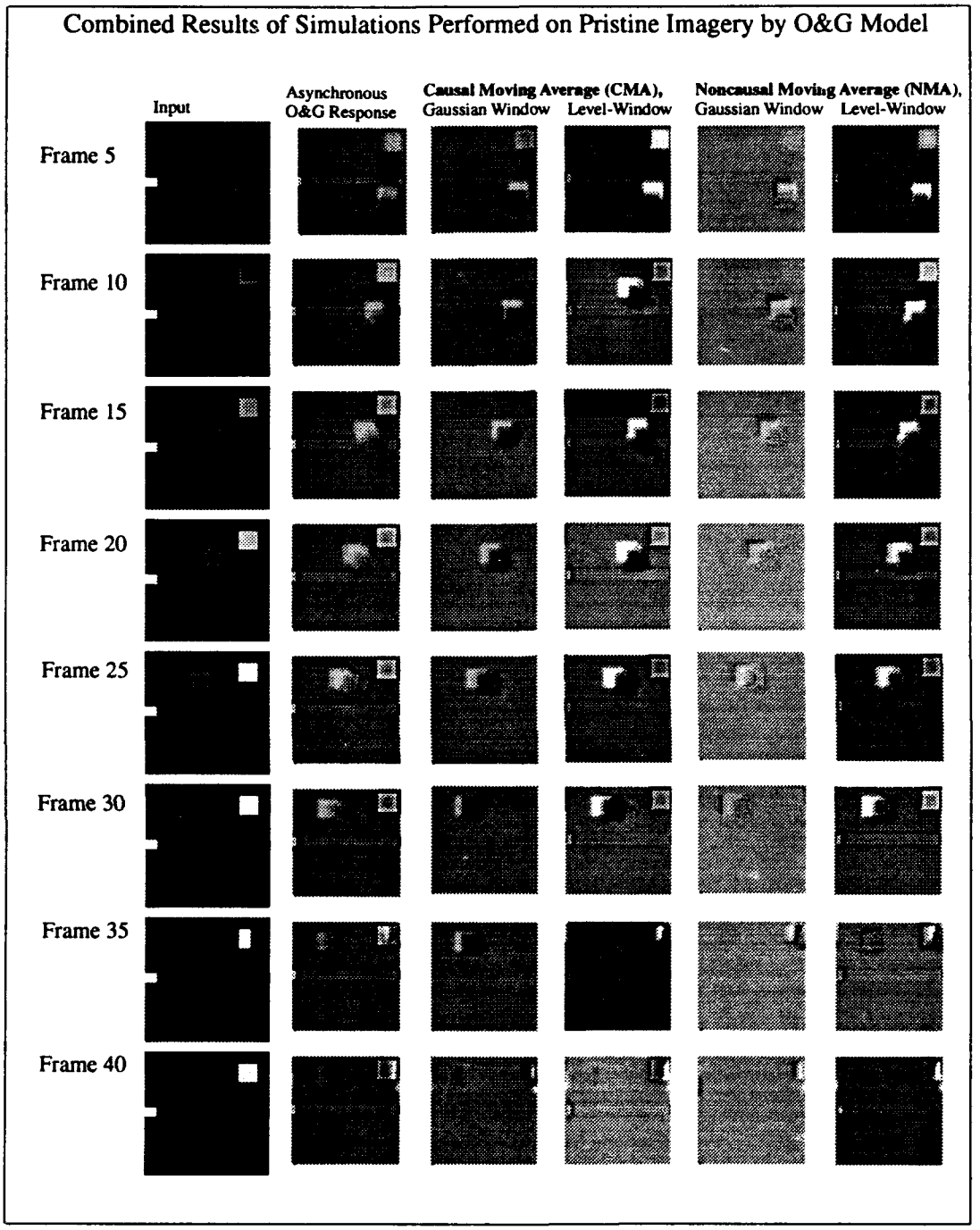

Figure 4.2. Pristine Simulation Results. This figure aligns every fifth image in the input sequence with the corresponding image from the O&G model output and two types of moving average algorithms. The input images are on the far left followed by the model response in the second column. The third and fourth columns contain the result of causal moving average post processing using either a gaussian-window or level-window weighting. The last two columns repeat the moving average processing using the noncausal filter.

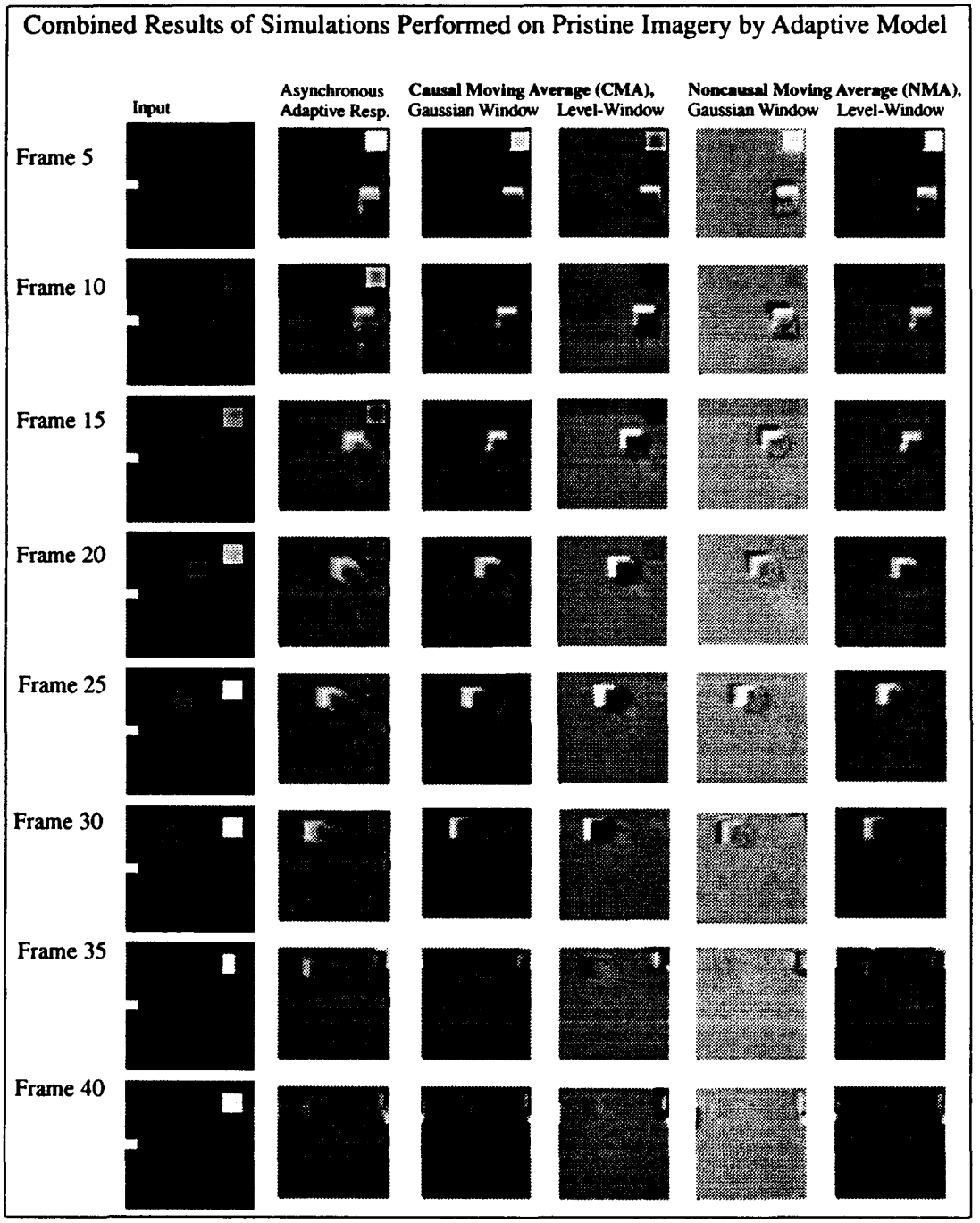

Figure 4.3. Pristine Simulation Results. This figure aligns every fifth image in the input sequence with the corresponding image from the adaptive model output and two types of moving average algorithms. The input images are on the far left followed by the model response in the second column. The third and fourth columns contained the result of causal moving average post processing using either a gaussian-window or level-window weighting. The last two columns repeat the moving average processing for the noncausal filter.

4.1.1.4 A Closer Study of Frame 10. Figure 4.4 has two rows of three frames. They corresponds to the model input (left), the O&G response, and the adaptive response (right). We

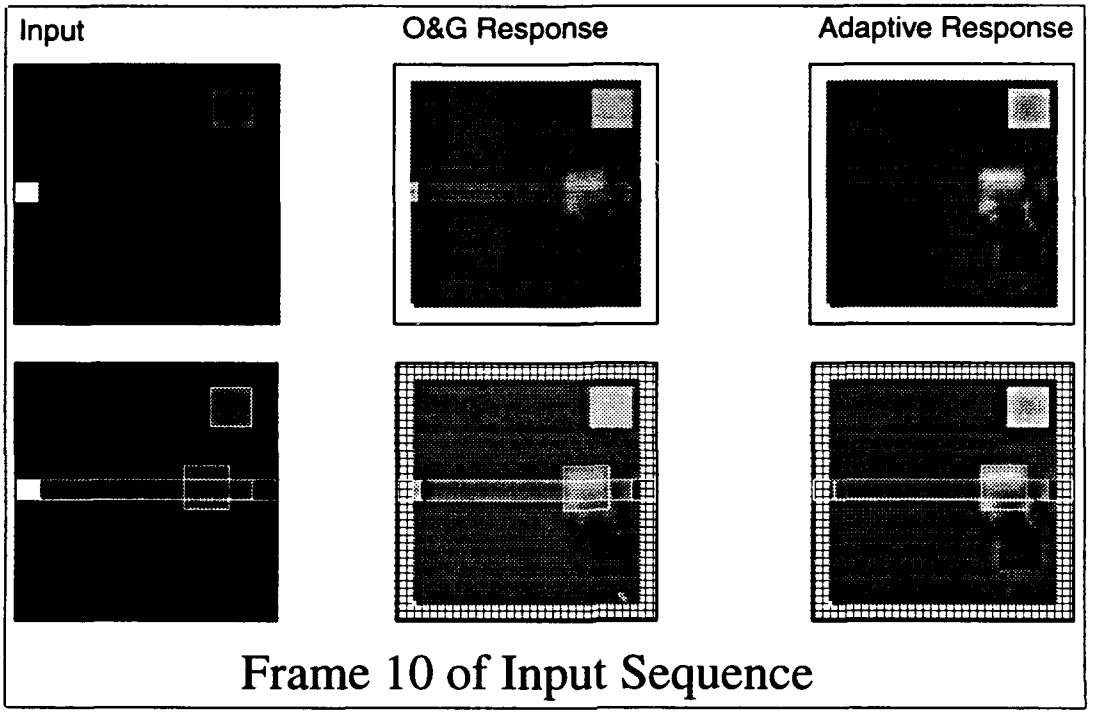

Figure 4.4. Pristine Simulation Frame 10 (Magnified). The two rows in this figure corresponds to frame 10 of the pristine imagery simulation. The frames are labeled as input (left), O&G response (middle), adaptive response (right). The bottom row is augmented with a white outline of the three salient features of the simulation - the moving box, the oscillating box (upper right), and the bar. The top row is the original frame. This frame shows the point where the moving box crosses the bar.

took the top row, the original frames, and created a row of frames with the location of the input features identified with a white outline (the bottom row). This allows us to pin-point the true location of the moving box at this time. It is interesting to see that in the O&G response, the bar has a faint, dark line running the length of the bar. An enlargement of this frame is provided in Figure 4.5.

Under greater magnification, we clearly see the dark lines running on either side of the bar. We also see that the hot end of the bar is surrounded by a nearly circular dark region. The cold end shows no sign of any perimeter activity. Returning to the moving box, we can faintly see the

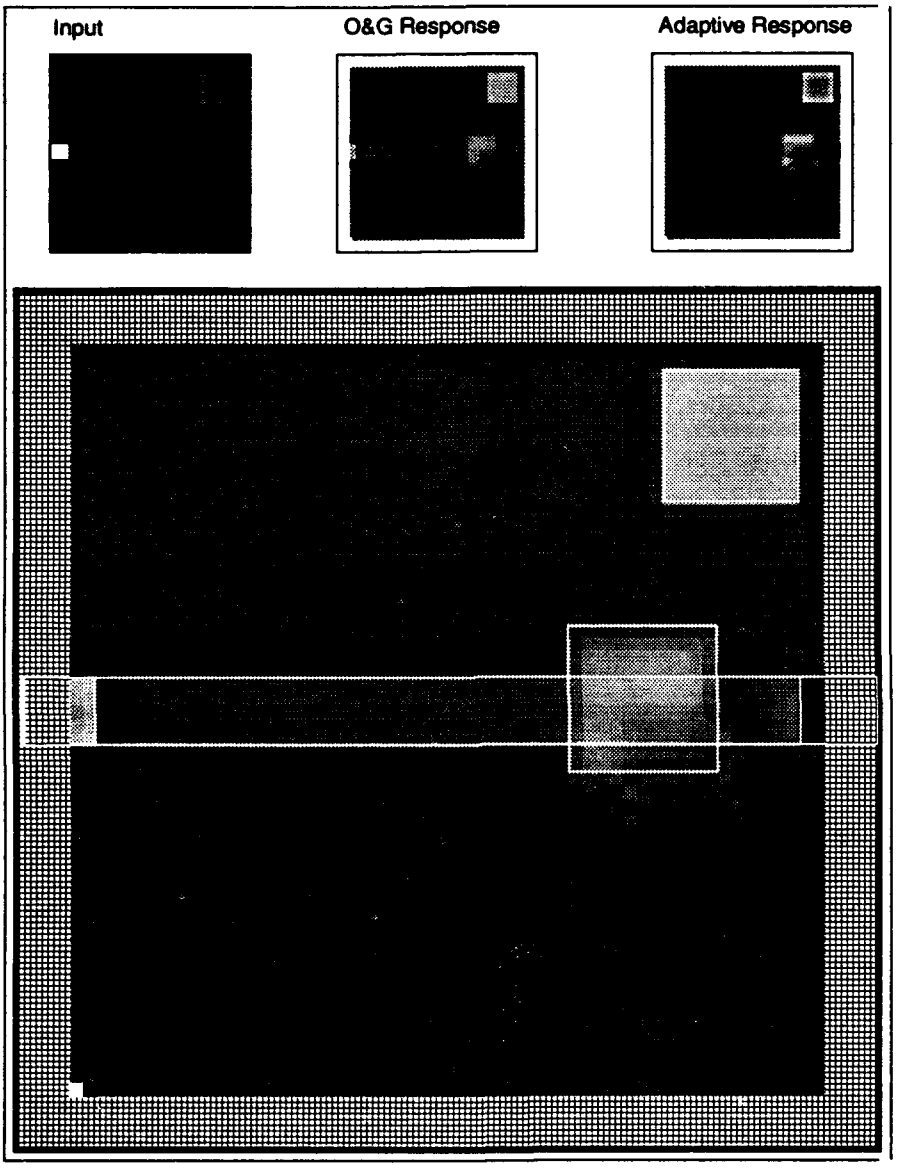

Figure 4.5. Pristine Simulation Frame 10 (O&G). This figure shows the three frames given in the top row of Figure 4.4, below which is the enlarged version of the O&G model response. The enlarged frame clearly shows the response of the model to the hot end of the stationary bar (left-center). A faint after image is apparant stemming from the moving box to the lower right. The black and white squares in the lower left corner are used to normalize this frame to the other frames in the sequence for display purposes. These squares are not apart of the model response.

dark line of the bar extending into the left side of the box region but it disappears due to the over riding response of the boxes *after-image*. The after-image is the shadow that extend behind the box like a wake following a ship. This after image is relatively short in comparison to that of the adaptive model. Figure 4.6 is an enlarged view of the adaptive response.

In the enlarged view of the adaptive model response, frame 10, we see the extension of the dark line into the moving box region as well as a lighter line extending into the cold region of the bar which is a complete reversal from the feature found in the O&G model. Additionally, the hot end shows no perimeter activity. Both the oscillating box and the after-image are pronounced in this figure compared to the O&G response. The time response of the adaptive model is so slow that the original location of the moving box is still rebounding. In Figure 4.6, we see there is a definitive drop in intensity along the bar where the box passes it. One last item to note, the edges of the oscillating box are brighter than its center or the outer perimeter.

The pronounced features of the adaptive model are all the result of having input nodes characterized by the long-term memory equation. Although the O&G model is quicker to respond to changes in input, it will not remove stationary elements as well as the adaptive model. We can help the O&G model to remove these features by modifying the output with a moving average algorithm. We discuss the results of post processing the response in the next section.

4.1.1.5 A Closer Study of Frame 15. If we look at frame 15 instead of 10, we will see the model has two moving edges. The response to these edges is interesting and so we use frame 15 in this discussion of moving average results. Figure 4.7 shows the magnified views of the response of our two models. Like Figure 4.4, we outlined the current location of the input features. We wouldn't know where the leading edges were from this figure alone, however, after processing the image with a moving average, we immediately see the leading edges. Figures 4.8 and 4.9 show the results of a gaussian-window and level-window moving average.

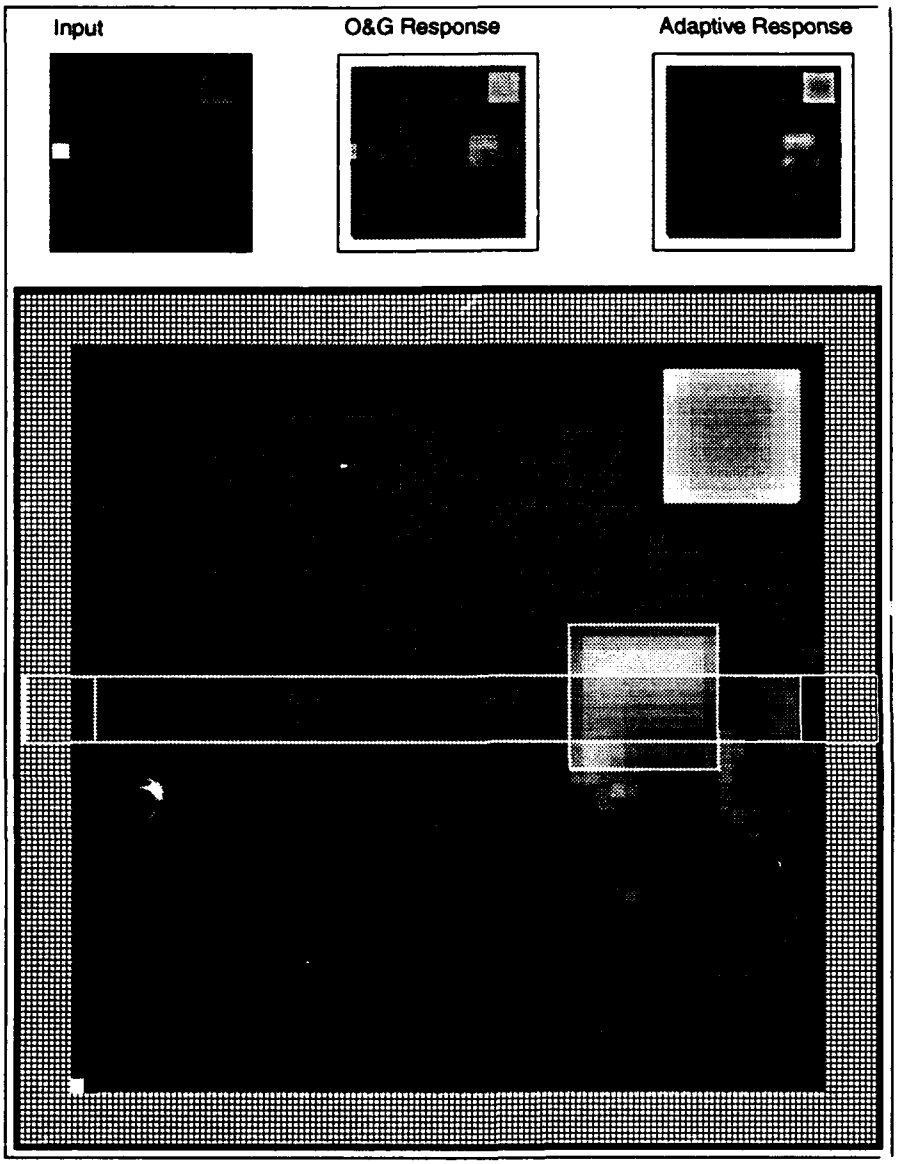

Figure 4.6. Pristine Simulation Frame 10 (adaptive). This figure shows the three frames given in the top row of Figure 4.4, below which is the enlarged version of the adaptive model response. The enlarged frame shows the faint response of the model to the cold end of the stationary bar (right-center). At the location where the box crosses the bar, the intensity is reduced below the box intensity but remains above the bar's intensity. This provides an indication of where the bar is even when the box crosses it. A strong after image is apparant stemming from the moving box to the lower right corner where the box orignated. The model responds to the oscillating square with a perimeter trough and an interior spike (upper left). The black and white squares in the lower left corner are used to normalize this frame to the other frames in the sequence for display purposes. These squares are not apart of the model response.

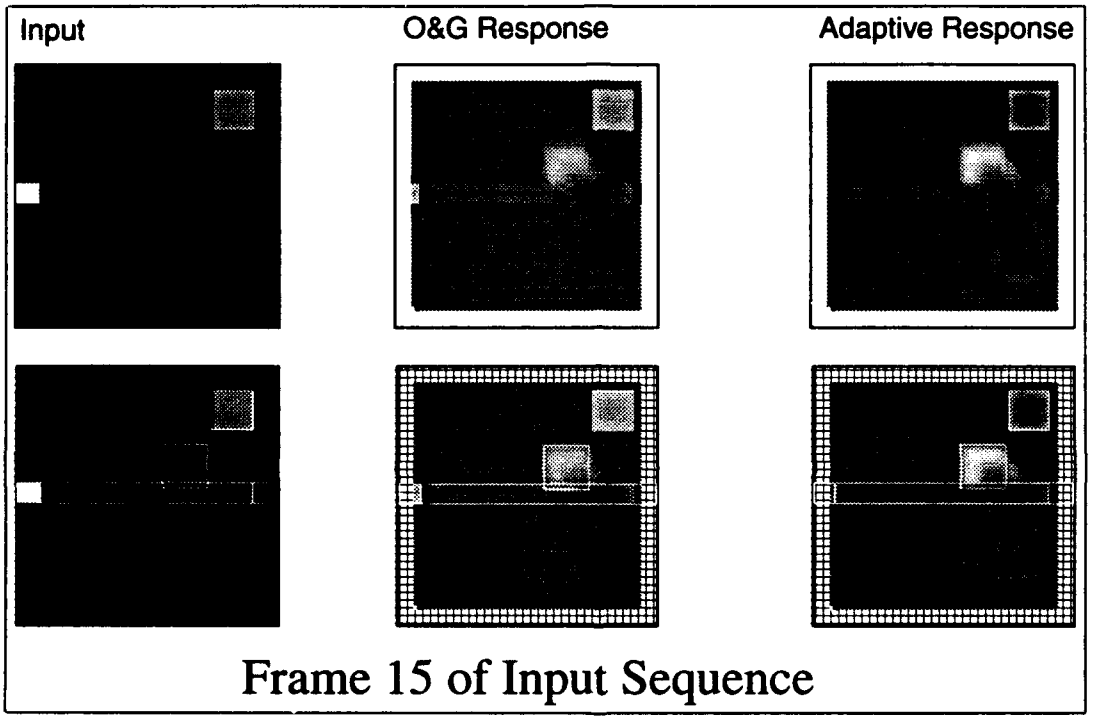

Figure 4.7. Pristine Simulation Frame 15 (Magnified). The two rows in this figure corresponds to frame 15 of the pristine imagery simulation. The frames are labeled as input (left), O&G response, adaptive response (right). The bottom row is augmented with a white outline of the three salient features of the simulation - the moving box, the oscillating box, and the bar. The top row is the original frame. This frame shows the a moving box with the leading edge extending across the left and top edges of the box.

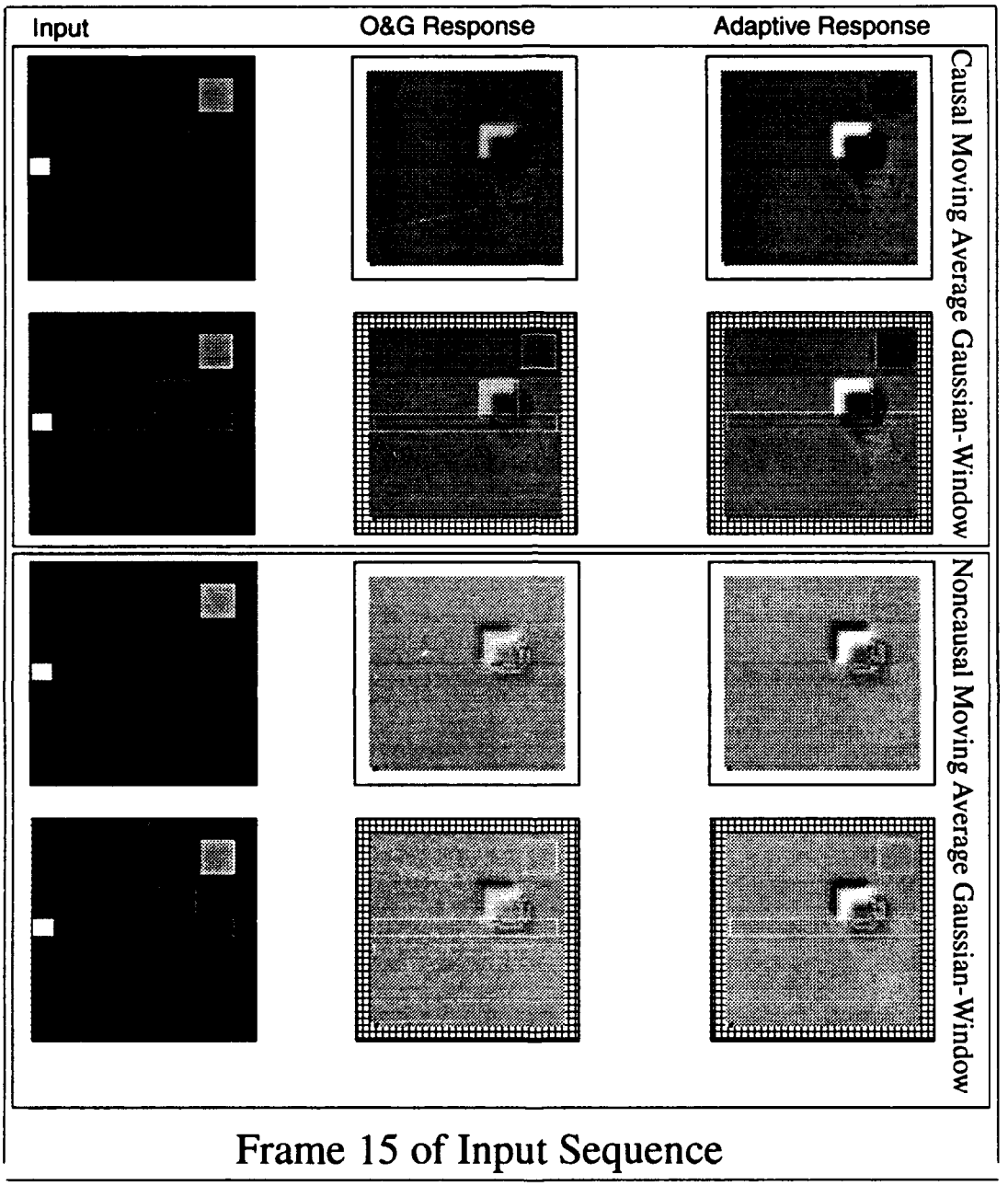

Figure 4.8. Pristine Simulation Frame 15 (Gaussian-Windowed). This figure contains two sets of frames after processing through the gaussian-windowed moving average algorithm. In both sets, the top row is the untouched versions of the input (left), the O&G response, adaptive response (right). The bottom row has a white outline identifying the location of the input features. The difference between the upper and lower set is the type of moving average used – causal (top set) and noncausal (bottom set).

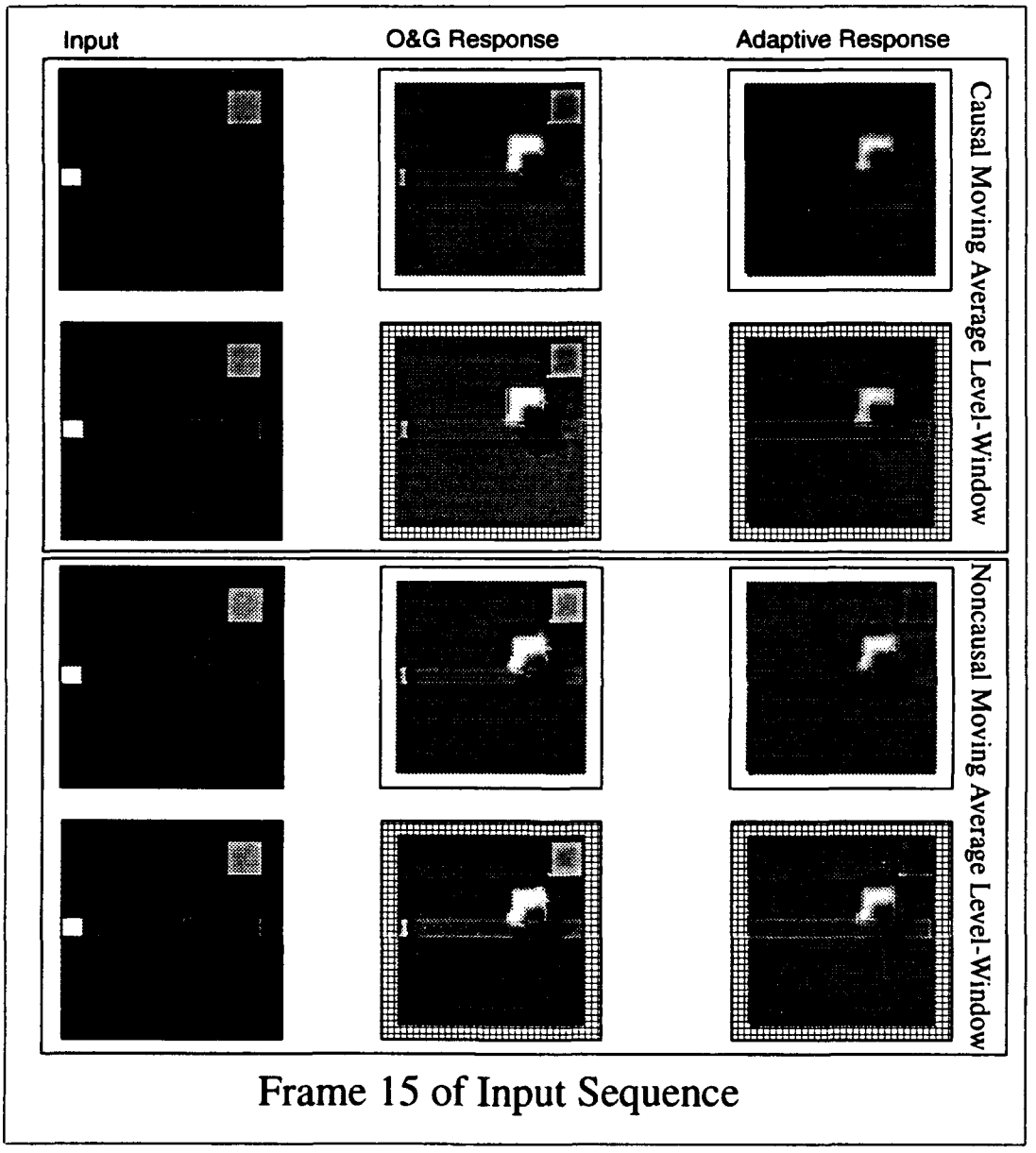

Figure 4.9. Pristine Simulation Frame 15 (Level-Windowed). This figure contains two sets of frames after processing through the level-windowed moving average algorithm. In both sets, the top row is the untouched versions of the input (left), the O&G response, adaptive response (right). The bottom row has a white outline identifying the location of the input features. The difference between the upper and lower set is the type of moving average used - causal (top set) and noncausal (bottom set).

These figures highlight different features. In the gaussian-windowed output, we see that the bar (including the end regions) has been completely removed. In the noncausal set of frames, the oscillating box is also gone leaving only the moving box. The left and top edges of the moving box have a dark region to the upper left followed by a bright area and then a fuzzy area indicative of an after image. The dark region identifies a trough and the bright area, a spike. These frames clearly indicate motion from the lower left to the upper right. We would expect this as the moving average adds information from frames outside the current time sample. In the level-window results, we still see the bar (clearly in the O&G response and very faintly in the adaptive response) and the oscillating box. The level-windowed frames have enhanced features. The dark areas are much darker than those in the gaussian-windowed results as are the bright areas. The after-images are still clear in these level-window results.

Of the eight possible combinations (four causal and four noncausal), we would select the noncausal, gaussian-windowed, moving average of the adaptive response as the *best* for identifying motion and direction while rejecting stationary features. We conclude this half of the chapter with the results of thresholding on the gaussian-window moving-averaged output, Figure 4.10.

We present this output to show the dramatic differences obtained under different thresholding schemes. The only comment we have on our thresholding results is that they depend on the threshold and type of thresholding. Therefore, any conclusions based on thresholding can not be made in general. In order to remain consistent, we present the magnified versions of frame 15 before moving on to processing FLIR imagery with our models.

4.1.2 Summary. In this half of the chapter we concentrated on analyzing the model response to pristine imagery. We saw that the O&G model was quicker to adapt to the changing input but was not able to remove stationary objects in the FOV. The after-image produced by the O&G model was shorter than the adaptive model. However, only after using the gaussian-window, moving-average filter on the O&G model output do we see the removal of stationary features. The

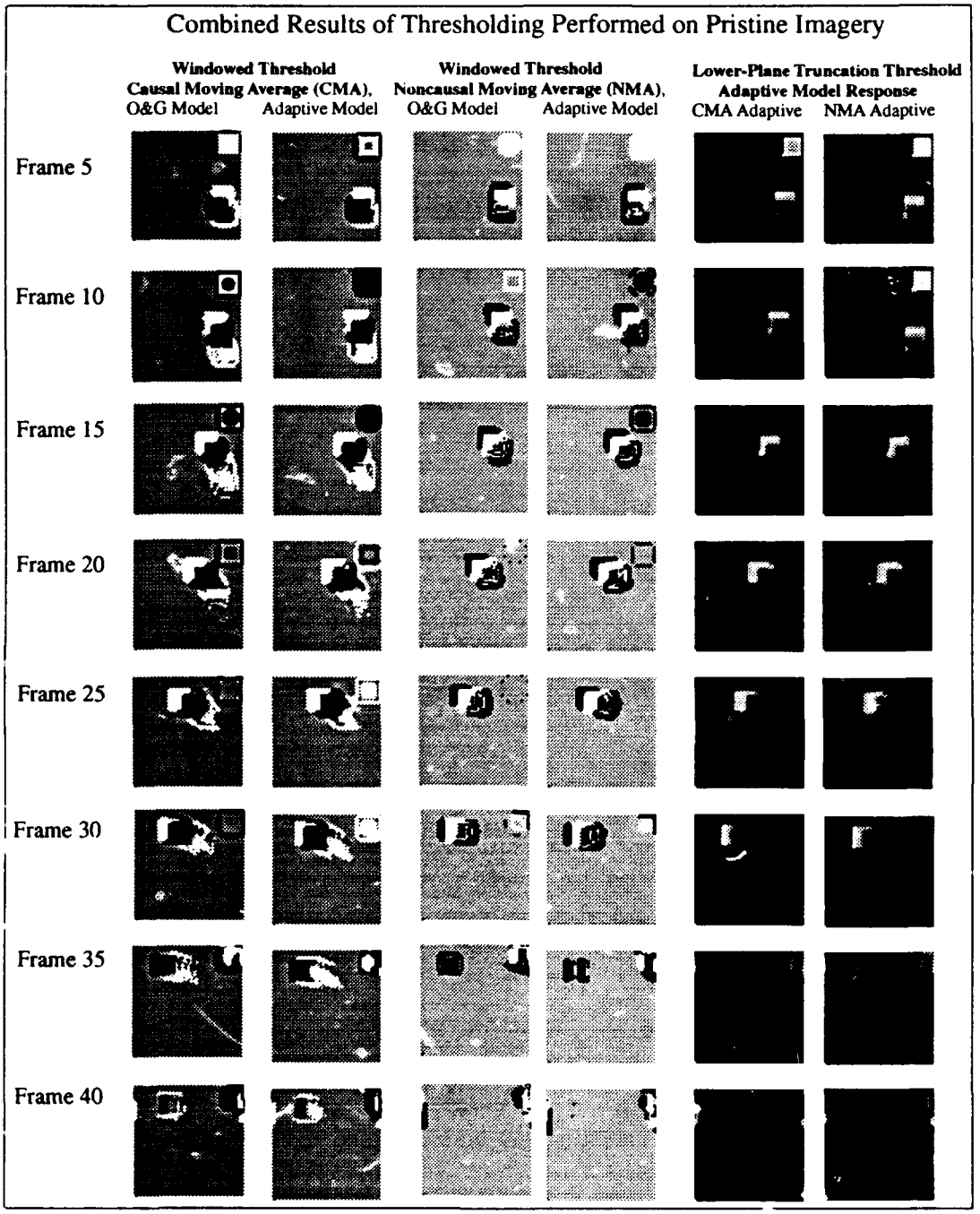

Figure 4.10. Pristine Simulation Threshold Results. This figure contains a synopsis of thresholding the gaussian-windowed output. The threshold of 0.1 is used for these figures. In the left four columns, we used a windowed threshold. All values above the threshold and below the negative threshold are set to their maximum or minimum values (respectively). This technique captures the zero crossing values only. The right two columns use lower-plane truncation. The values lower than the threshold are set to the threshold.

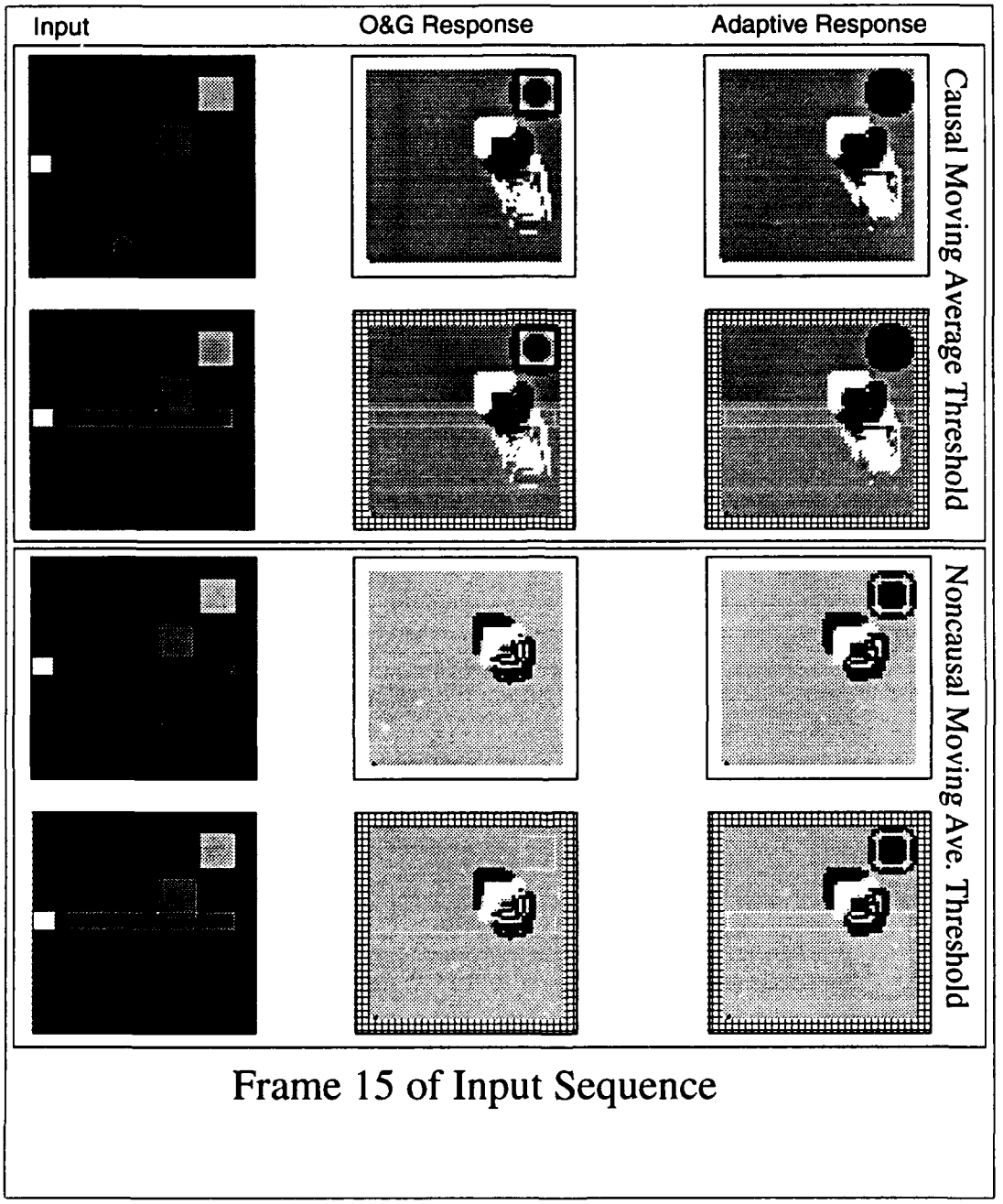

Figure 4.11. Pristine Simulation Frame 15 (Gaussian-Windowed and Thresholded). This figure contains two sets of frames after processing through the gaussian-windowed moving average algorithm and threshold routine. In both sets, the top row is the untouched versions of the input (left), the O&G response, adaptive response (right). The bottom row has a white outline identifying the location of the input features. The difference between the upper and lower set is the type of moving average used – causal (top set) and noncausal (bottom set).

adaptive model, on the other hand, was far more effective at removing stationary objects. This model's response could be post processed by either the level-window or gaussian-window moving average to further remove stationary images.

In comparing the causal and noncausal post process, both have aid in removing stationary features, but only the noncausal filter was able to produce a leading edge with a forward dark region (trough) followed by a bright region (spike) straddling the object's edge. It is this trough/spike combination that identified the direction of motion of the object and can only be obtained by combining past and future values into the current image. In the next half of this chapter, we look at our models using real Forward Looking Infrared Radar (FLIR) image sequences.

4.2 Simulations Performed on FLIR Imagery

4.2.1 FLIR Generation and Description. The input sequences obtained for simulations made using FLIR imagery were captured using a Trapix 4B Virtual Image Processor. The original imagery was recorded onto a VHS video tape. The sequences were captured at a rate of 32 frames a second. The actual size of each image was 700 by 512 pixels (columns by rows). We captured just over eight minutes of video giving us some 260 frames in the sequence. Because of the processing time involved with running our models, the images were cropped to 256 by 128 pixels (an eight fold increase in the size of our input FOV). The sequence focused on a tank moving in the images. Figure 4.12 illustrates how we isolated the tank and created a smaller, cropped version of each frame. The frames below the cropping diagram illustrate the first frames produced by the adaptive model. We will return to these frames later.

For most of the sequence, the FOV remains stationary. However, in the last 60 frames, the camera follows the tank as it moves out of its FOV. A full description of the steps taken to process this imagery is provided as Appendix B. In this section we will look at the response of our

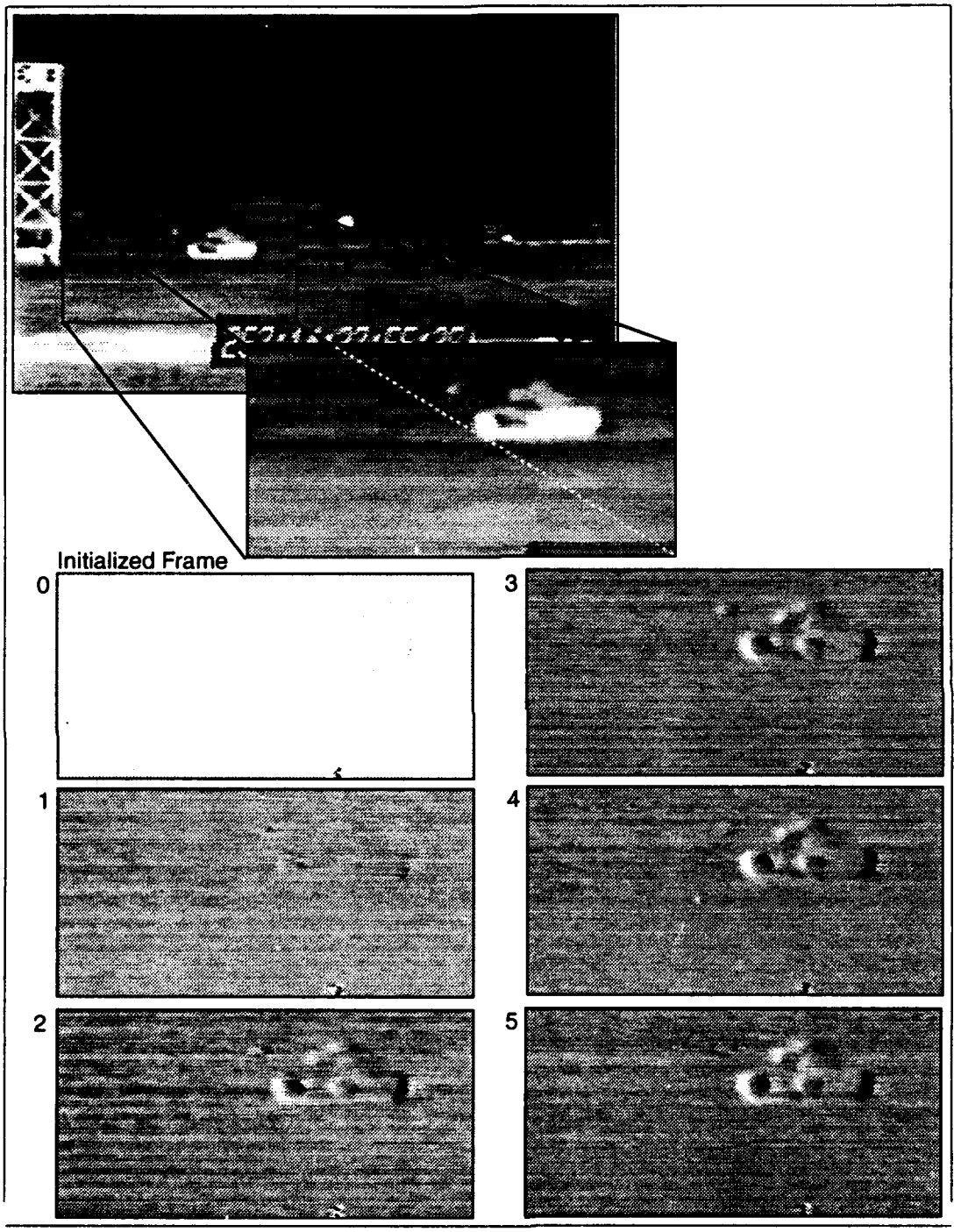

Figure 4.12. This figure shows how the images used as input to our model were extracted from imagery with a much greater FOV. The original frames, marked by a tower along the left edge, were 700 pixels wide by 512 pixels tall. The cropped image is 256 pixels by 128 pixels. Below the cropping figure are the first six frames produced by the adaptive model. Here we see the initial frame is totally adapted to while subsequent frames show the appearance of a tank which is in motion.

Table 4.3. Parameters used to configure the simulation software for the tank input sequence.

Synapse Parameters		
α	0.5	Natural Decay Term
β	5.0	Target Value
SYNTOL	0.0005	RungeKutta Tolerance
Soma Parameters		
A	5.0	Natural Decay Term
B	45.0	Upper Bound
D	45.0	Lower Bound
SOMATOL	0.0005	RungeKutta Tolerance
TIME_DELAY	0.05	Inhibitory Delay
Neuron Size		
Receptive Field	9 Nodes	Vertical Direction
	5 Nodes	Horizontal Direction
Excitatory Region	5 Nodes	Vertical Direction
	3 Nodes	Horizontal Direction

model to the stationary and moving FOV. We use both the O&G asynchronous models followed by gaussian-windowed, moving-average filters.

In order to apply our models to the tank imagery, we had to expand the model's FOV to 256 by 128. We also configured the receptive field to be a rectangle with nine nodes in the horizontal direction and five nodes in the vertical. The excitatory region was appropriately scaled to five horizontal nodes and three vertical nodes. Table 4.3 lists the values used in these simulation. Before getting into the model performance, we need to note that the tank imagery had values between zero and 255 (gray-levels) but our model was built and tested using values between zero and 16. Therefore, we divided the tank image pixel values by 63.75 to reduce them to values more inline with our other simulations.

4.2.2 Model Performance. The image sequence can be divided into two phases. The stationary FOV phase and the moving FOV phase. We look at the response of our model and the results of post processing in both phases beginning with the stationary phase.

4.2.2.1 Stationary Field-of-View Segment. The sequence we chose to use has relatively few objects in the FOV. Figure 4.13 shows how the tank moves from the right to left, turns and move off to the right. Two interesting features are evident in these images. In the lower 12 rows of

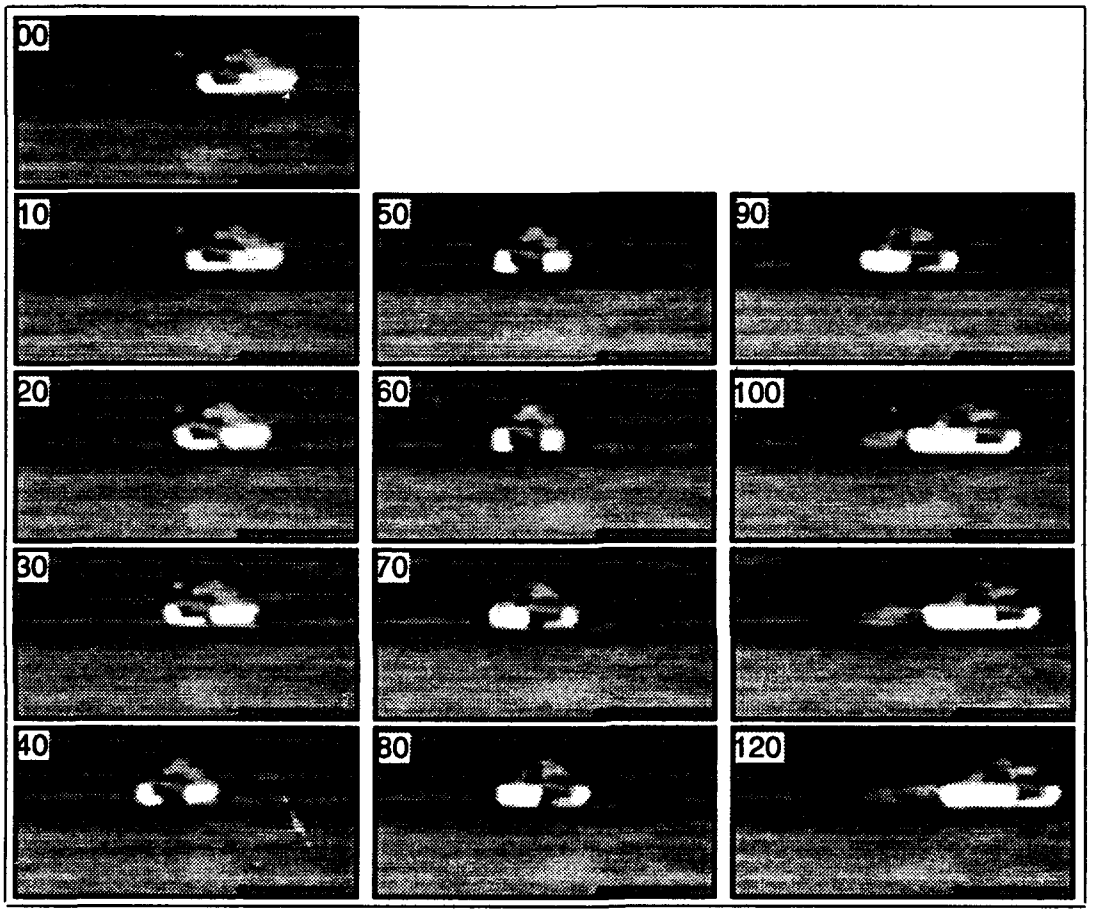

Figure 4.13. The images shown here are selected to illustrate the period that the FOV is stationary. The tank is clearly visible throughout the sequence. Frame numbers are posted in the upper left corners.

each frame is a black speck flanked by a homogeneous band running to the right of the frame. This is the time code block placed into the frame by the FLIR camera. The black speck oscillates at an extremely high frequency throughout the video. We will see the effect of this artificially generated feature later. The second feature to note is the plume of exhaust trailing the tank. In frame 100 we see it first. The plume moves slowly and to right as the tank speeds off the frame. We will look for this as we examine the model response.

In the following discussion, we look at the model outputs starting with frame 10 and every tenth frame thereafter. We skip the early frames because the model starts by adapting totally to the first frame. This results in a blank image and can be seen in the frames included in Figure 4.12. The first five frames show the tank slowly appearing with time. The time code speck is apparent even in the initial frame because it oscillates at an extremely high frequency and represents a narrow, large value edge which even our model can not ignore.

Figure 4.14 shows the response of the O&G model to the tank imagery.This figure shows how

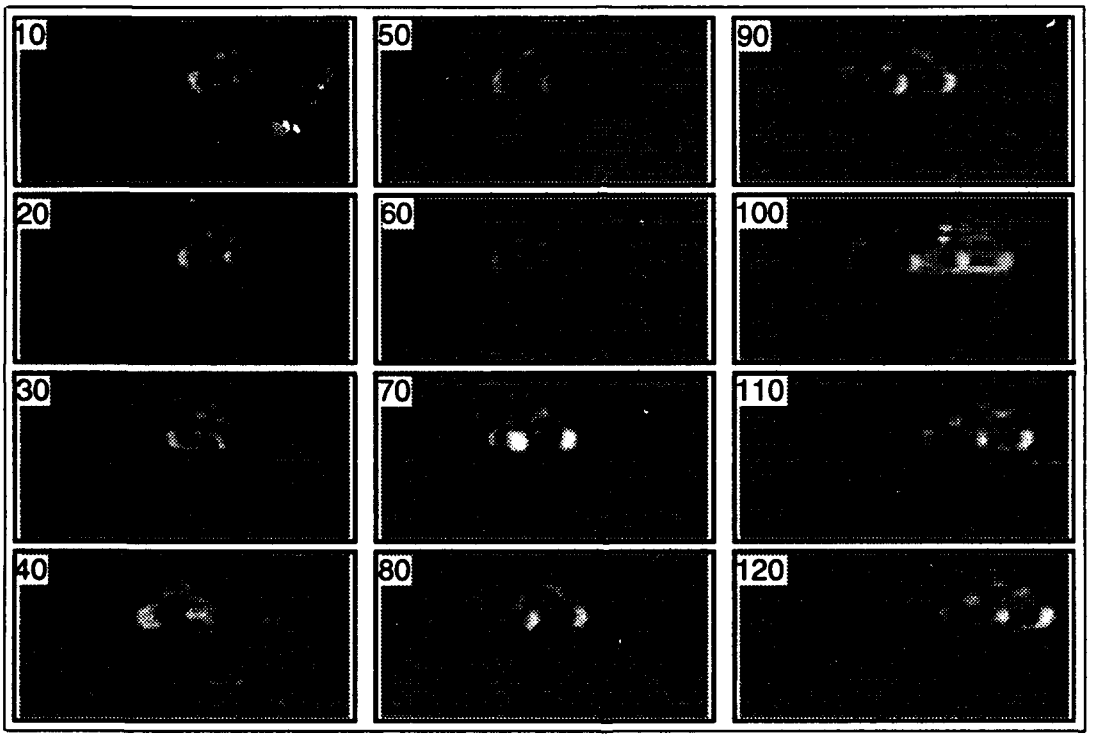

Figure 4.14. The O&G model response to the tank input sequence.

the model smooths the background plane to a nearly uniform field. The tank is clearly visible in these frames as is the time code speck. We see in frame 50 and 60 that the tank, during its turn, is less pronounced than in other frames. This is because the tanks lateral velocity has decrease significantly even though the tank is moving the same speed throughout the simulation. We see that the leading edges of the tank is highlighted with a bright region and the trailing edges have dark regions behind them.

In frame 70, the leading edges are the brightest of all the frames in the sequence. This is because the leading edge moves into the former trailing edge region. Therefore, the appearance of motion in a region trying to rebound causes the model to be excited to a greater level than seen when the edges moves into a region at the ambient level. We see in subsequent frames that the leading edge returns to a *normal* level. This feature is indicative of an object reversing direction.

In frame 100 there is an extended after-image. This is the effect of skipping frames in the input sequence giving the model the impression that the object changes speed nearly instantaneously. The new location of the object is almost all outlined by a bright region and the old location is a dark shadow. Frame 40 shows some of this behavior as it too follows a jump in the input sequence. If we look closely, we can see the exhaust plume trailing in frames 90, 110, and 120.

Figure 4.15 is the same frames from the adaptive model response.The same comments can

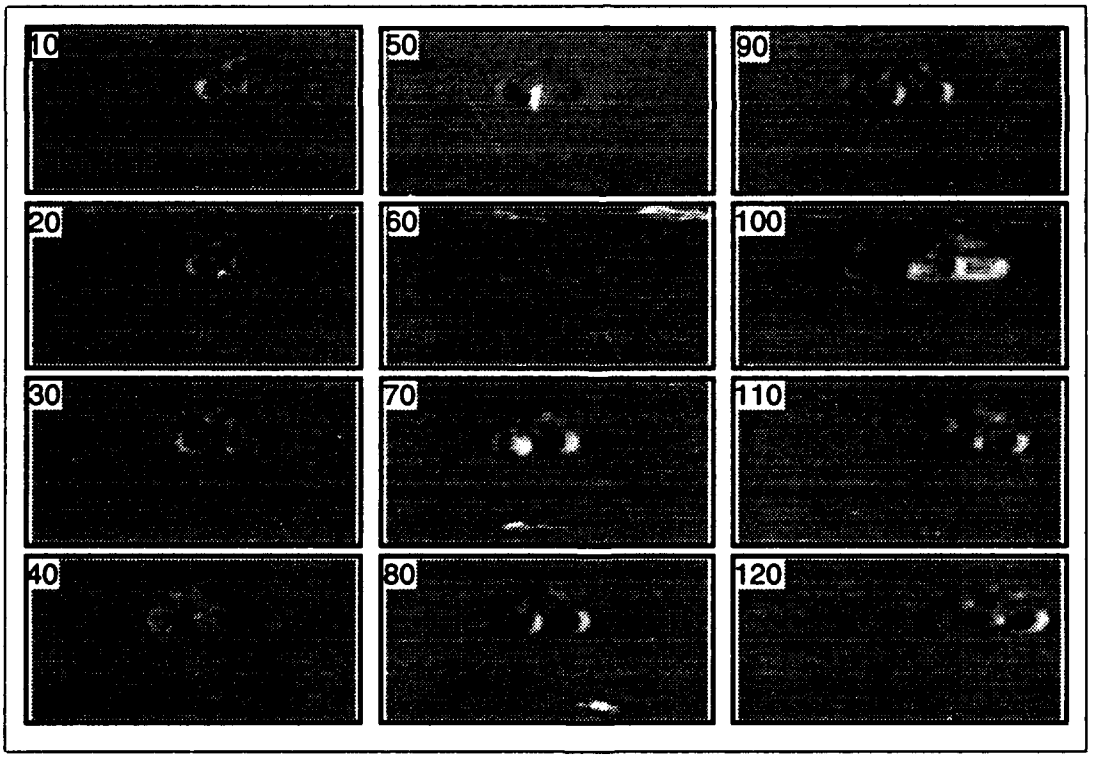

Figure 4.15. The adaptive model response to the tank input sequence.

be made about the frames as were made with the O&G model response. The difference is the loss

of sharpness in the images. The adaptive model is less distinct than the O&G model. This is best illustrated by the nearly lost exhaust plume in frames 90 - 120. We will investigate the second phase of the sequence next.

4.2.2.2 Moving Field-of-View Segment. Figure 4.16 shows the sequence of images in which the FOV is moving. The FOV is actually moving faster than the tank in order to place it back in the center of the FOV. The FOV moves in frames 200 through 225 and then stops. During the 25 frames of motion, the leading edge of the tank is on the left which is the edge of relative motion to the FOV. When the FOV stops, the leading edges switch to the true front of the tank. In the model responses, we see that all the objects in the FOV are identified as moving. When the FOV stops these objects disappear, leaving only the tank. Notice in frames 220, the post below the tank in the O&G model is blurred while the adaptive model shows a well defined post with a leading and trailing edge. In this case, the adaptive model shows crisper edges. As with our previous simulations, we will post process the imagery with a moving-average filter.

4.2.3 Results of Post-Processing Techniques. When we processed the adaptive model's response through a gaussian-windowed, moving-average filter, we were surprised to see no improvement of the images. In Figures 4.17 and 4.18 we show the causal and noncausal filter output. The images produced by the causal filter increased the response of the tank's moving edges while suppressing the surfaces in between the edges. At the price of losing the outline of the tank, we gained more information on its motion. Consistent with the pristine image results, the leading edge is identified by a bright region followed by a dark region. In frames 50, 60, and 70, the tank is turning and its relative, lateral motion is nearly zero. In these frames we nearly lose the tank entirely. In frame 80 and 90 we see the front edges of the tank's tread as leading edges and know immediately that the tank is moving to the right.

The images produced by the noncausal filter are extremely fuzzy. We do, however, see the trough preceding all the leading edges of the image. Frame 110 has the best example of this feature.

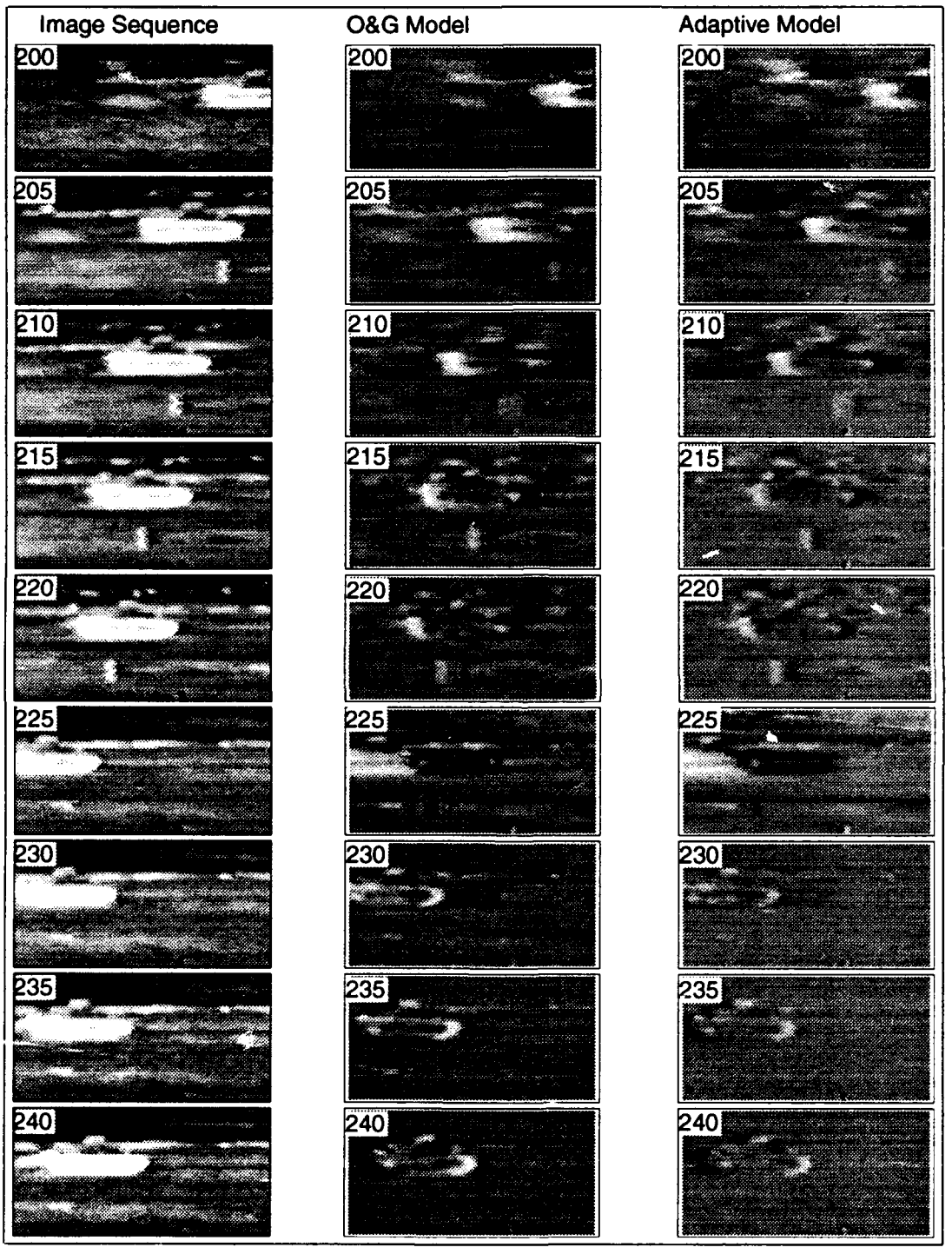

Figure 4.16. Here there are nine frames space five frames apart showing the period of motion in our tank imagery. The input frames are shown on the left, the O&G model response is in the center column, and the adaptive response is on the right.

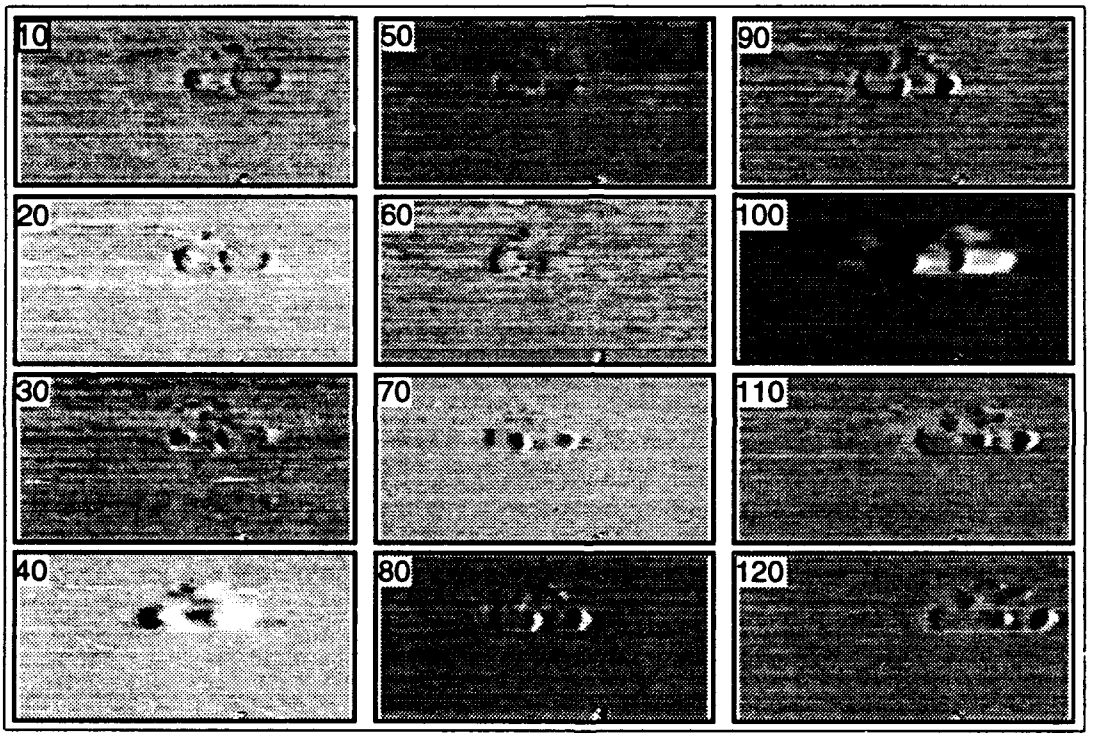

Figure 4.17. The adaptive model response to the tank input sequence is processed through a moving-average filter. These frames show the results of a causal moving average.

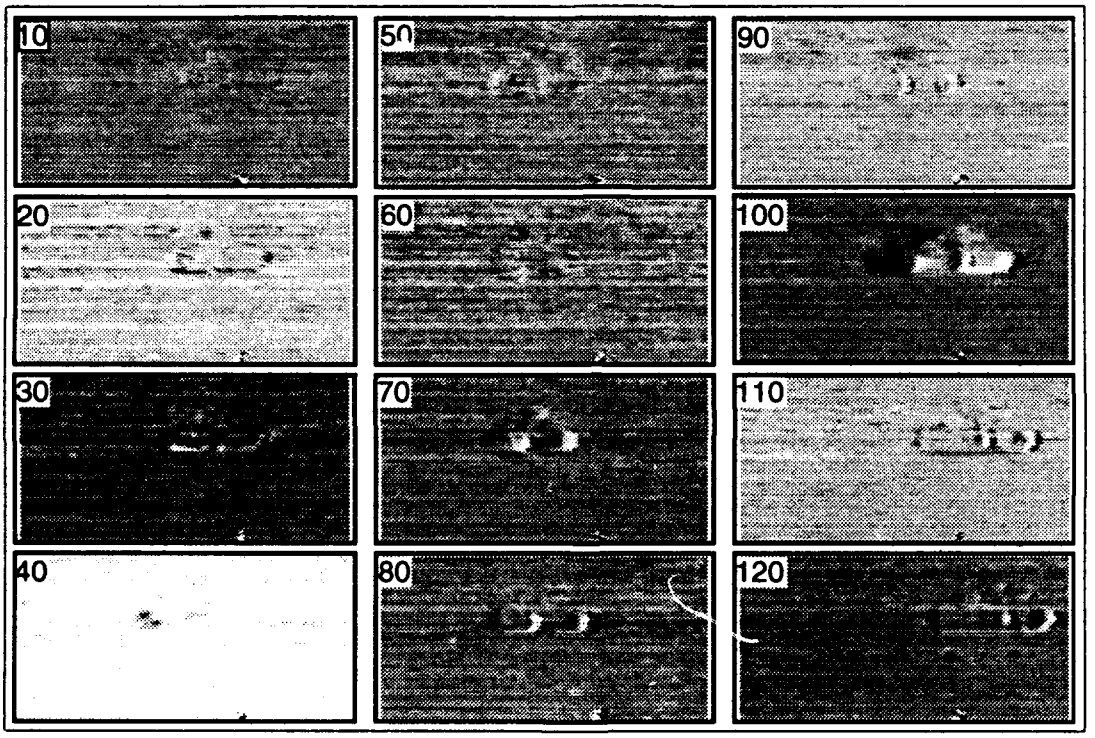

Figure 4.18. The adaptive model response to the tank input sequence is processed through a moving-average filter. These frames show the results of a noncausal filter.

A sharp, dark edge outlines the bright spike of the edge's response. In these frames, we see that the tank is moving first to the left and then to the right with almost no motion in frames 50 through 70. We will not attempt thresholding on these images because we are not convinced that thresholding techniques can be implemented in a general way.

4.3 Summary

In this chapter we concerned ourselves with applying our models to two-dimensional image sequences. The pristine images allowed us to investigate the behavior of our models without the presence of noise. We saw that the O&G model was a faster model and was able to follow the moving object with only a small after-image. The adaptive model, on the other hand is a slow model and produces a significant after-image. However, the adaptive model was able to reduce the response of stationary objects which the O&G model did only poorly. The oscillating box was also reduced significantly in the adaptive model while the O&G model always had a shadow of it in the output FOV. We saw that the two models had opposite effects on the vertical bar's ends. The O&G model highlighted the bright end while ignoring the dark end. The adaptive model highlighted the dark end and ignored the bright end. The output of both models could be altered by the use of post filtering using a moving average.

We applied two types of moving-average windows to our model outputs and found the causal, level-windowed filter to be the least effective and the gaussian-windowed, noncausal filter to have the more favorable effect. This latter filter was able to identify the direction of the moving square by a deep trough leading the edges forward wave. We concluded that this was to be expected as the noncausal filter uses data from the future to determine the present output and will be able to show future position in the present output. We applied the models to real FLIR imagery in the second half of the chapter.

The FLIR imagery was full of noise but our model was able to extract only the moving tank in the FOV as long as the FOV was stationary. As soon as the FOV began to move, the entire scene was identified as moving. A result not wholly unexpected. The interesting aspect of the moving FOV was that the tank's relative, lateral motion was highlighted until the FOV became stationary once again. We attempted to apply the moving-average filters to our model output but found that the filters blended the tank into the background and only faint features located at the tank's edges remained. The noncausal filter maintained the trough leading the spiked edge feature indicating the direction of motion but this was difficult to detect. We compare our results with the objectives of our research in Chapter V as well as answer our research questions explicitly.

V. Summary and Conclusions

In our research we attempted to model the motion sensitive neurons of the retina in order to better identify moving objects in a scene. We based our work on that of H. Öğmen and S. Gagné concerning the fly's visual system. In examining their models, we noticed no mention of a network which uses both the cell-activity and neurotransmitter equations to form a single neuron. This became the center of our thesis work – to investigate the differences in the O&G model and our newly defined adaptive model. We used software designed outside the scope of this research to simulate the model architecture. We began by applying input sequences to a linearly defined (one-dimensional) model. In doing so, we were able to gain confidence in the model's response and verify that it operated as described by H. Öğmen and S. Gagné. The linear response graphs gained from these early simulations prepared us for investigating simulations on a two-dimensional input sequence. We concentrated on the asynchronous model for its ability to heighten the model's response to transient behavior. Throughout this thesis we used the asynchronous model and little was said about the synchronous models.

The O&G and adaptive models were applied to pristine images in which the input scene was noise free and had three interesting features – motion, temporal intensity changes, and stationary objects. We found that both the models performed well in different areas. The O&G model was quick to follow the input but did not reduce the response of stationary objects. The adaptive model did a better job of reducing stationary objects but was slower in is ability to rebound from an adapted state. This left an extended after-image in the output images. We were able to remove the after-image and most of the stationary images by using a moving-average filter. The two types examined were the gaussian-windowed and the level-windowed filter. We found the gaussian-windowed filter to provide higher rejection of stationary items and in the noncausal design, was able to highlight motion direction on the leading edge. Our attempt to threshold the

output images resulted in mixed findings and we were unable to confidently make any conclusions to its effectiveness.

In simulations using the FLIR imagery, our models produced excellent output that identified only the moving tank in the image. We found that application of our filters aggravated the details of the tank image but in the noncausal case, did identify direction of motion. Our specific findings in accordance with our research objectives and questions are given in the next two sections. We conclude this thesis with a discussion of the application of the model.

5.1 Research Objectives

We identified five objectives in Chapter I of this thesis. In this section we will take each and comment on how each objective was achieved.

1. **Examine the model response in regards to leading and trailing edges, time-variant stimuli, and velocity and direction of motion.** In Chapter III we showed, using the linear models, that the trailing and leading edges could be uniquely identified in the output response of the model. The model's response to stationary, time-variant stimuli was difficult to interpret. In the simulations based on pristine imagery, we found that the direction could be identified rather easily using a noncausal filter. Determination of the input velocity was addressed in one linear simulation. We found the model could be tuned to different velocities.
2. **Compare the response of the two-dimensional models with adaptive and non-adaptive inputs.** The O&G model is a faster model. It is capable of following the input more closely than the adaptive model because it contains no long-term memory of the input. However, the long-term memory is the very reason the adaptive model is better at removing stationary objects in the input scene.
3. **Compare the response of our models to both the pristine and FLIR image sequences.** We found the models were well behaved when given the pristine imagery. The FLIR imagery, on the other hand, was full of high frequency background components and noise. Even so, we obtained results that distinctly identified the tank location and leading edges. We will return to this in the next section.
4. **Post process the response data of the models to enhance the transient portions of the images.** Our limited attempt to draw more information from the model output was extremely successful. We were able to enhance the model's response by using both the causal and noncausal filters.

5.2 Research Questions

We set out to answer six basic questions concerning the behavior of the O&G and adaptive models. In this section we revisit each and provide our findings.

1. **How do the models respond to pristine image sequences?** Both the O&G and adaptive model highlighted the moving square in the pristine sequences. They differed in their ability to respond to motion and their suppression of stationary features. When we coupled the adaptive model to a gaussian-windowed, moving-average filter, we found that the output could show the current leading edge and its direction.
2. **What are the major features of the model's response to these inputs?** We showed in Chapter III that the output from the linear models could be annotated with time dependent features and spacial features. The overshoot, undershoot, and rebound all changed over the time axis. The spike, trough, plateau, and mesa were all associated with a variation over the spacial axis. A combination of the spike and overshoot gives rise to the forward wave of the output indicating a moving, leading edge. The trough and mesa combined with the antagonistic rebound created a trailing wave which identified the trailing edge.
3. **Can the model operate on images acquired by Forward Looking Infrared Radar image sequences?** We saw in the last chapter that the model definitely could operate on FLIR imagery. In fact, it does an excellent job of identifying motion when the FOV remains stationary. We found that further post processing was unnecessary.
4. **What is the effect of moving the the image's field-of-view with a moving object?** A moving FOV translated into motion of all objects therein. We showed how this motion blurred the scene and the objects moving in it had varying responses based on their relative, lateral velocity.
5. **Can the output be further processed to better segment motion?** We saw in the pristine motion this was true. The FLIR imagery, however, looks like the noncausal filter was able to enhance moving elements.
6. **How do causal and noncausal filtering effect the model output?** The causal filter shows gradients of the output imagery based only on past information, while the noncausal filter adds future data to give direction information. The causal filter aided in removing stationary objects but the noncausal filter completely removed the objects and did a better job of eliminating the oscillating features. This filter provided an indication of where the motion would move to in subsequent frames as well.

5.3 Application of Model and Future Research

The model we developed could be used directly or in a cascade system designed to preprocess motion information. As a stand-alone preprocessor, the model can be used to identify moving objects in a sensor's FOV. The processing hardware might use this information to segment the objects from the scene and apply pattern recognition techniques to identify them. Not only does the model allow for image-by-image segmentation, but taken over time, the pattern recognition engine might use motion characteristics as an additional feature for better classification. Another use of the model is to allow for the fusion of two sensors with varying characteristics. As an example, the model might take sensor data from a low resolution system with a wide FOV (FLIR) and provide object positions to a high resolution system with a small FOV (laser radar). In this configuration, the FLIR might be implemented as a starring sensor while the laser radar would be a directed system. The model might also be integrated into a vision system such as those being

developed by Caltech or Amber Engineering. In this case, we need to understand where the model fits into the visual system and what work needs to be accomplished hereafter.

As we noted earlier, the first step to emulating the retina was taken by Carver Mead and Misha Mahowald at Caltech. Their Silicon Retina represents the retina's top three neural layers - photoreceptors, horizontal cell, and bipolar cells. Our adaptive model fits directly behind the bipolar cell and could be thought of as the amacrine or ganglion cell layers. Since the model performed better at segmenting motion in the pristine sequence than in the noisy FLIR imagery, we believe it important that the input to the model be as noise free as possible. This coupled with the results given by Mead and Mahowald indicate that the Silicon Retina or its cousin the Neuromorphic Infrared Focal Plane processor could be used as a front end to our model. However, because the adaptive model requires a moving-average filter to create directional information and the output nodes are sensitive to motion in any direction (not *directionally selective*), we must conclude that our model can not be the output layer of any simulated retinal design. Therefore, the adaptive model can only represent the amacrine cell layer. Further research into creating models for the ganglion cells should follow our findings. Models that provide directional information and can be used to drive decision-making neural networks.

Appendix A. **Extended Literature Review**

The information contained in this appendix represents the basic concepts required to understand our thesis effort. It is provided for those readers who have not the formal background in neuroanatomy, neural networks, and early vision research.

Early vision research can be divided into three fields. The first is the field of observation. Experiments are performed on the retina of various animals to observe its structural and functional makeup. In much of early vision literature, the pronoun *wetware* is used to identify physiological components. The second field is that of modeling. Models are formulated using math expressions, usually differential equations, to simulate the operation of components of the retina and compared against observed phenomena. The last field is hardware design. The ultimate goal of early vision research is to design hardware devices that incorporate functionally efficient aspects of the retina. A review of early vision research is the subject of this chapter.

A.1 Scope of the Review and Method of Treatment

When the author began this literature review of retinal models he had no understanding of neuroanatomy or the retina. Therefore, this chapter is scoped for someone without a background in neural networks or vision research. Someone who needs to understand the basic principles of vision in order to apply them to any of a number of problems related to modeling the human retina. Although a full treatment of the entire visual process is beyond our abilities, the following paper touches on the basic concepts and directs the reader to articles and authors currently working in the area of primate vision and retinal modeling. Building on the basics, we examine related research to provide a sense of environment. By expanding the scope of this review to include a slightly larger picture than absolutely necessary, the paper can be used to initiate newcomers to the study of retinal based models.

A.1.1 Organization. The body of the review is broken into two sections. The first deals with the neuron and the physical structure of the retina and gives detailed information about each of the five types of neurons used to form the retina. Concepts such as lateral inhibition, receptive fields, and shunting inhibition are all subjects of this section. The last section discusses current retinal emulators.

A.2 Vision in Man

A.2.1 "The Neuron" [Stevens, 1979]. This section reviews the morphology of the nerve cell or neuron. There are three distinct regions of the neuron. They are the cell body or soma, the dendrites, and the axon. Metabolically, the cell relies on the soma to provide enzymes and molecules essential to life. Functionally, however, the soma integrates signals received through tubelike extensions of the cell membrane called dendrites. Forming a dense plexus of interconnections, the dendritic tree receives signals from as many as 10,000 other neurons. If the received signal strengths are sufficient to elicit a response from the neuron, a pulse is sent down the long axon used to transmit information from the cell to the brain or body. Figure A.1 depicts a typical neuron.

The neuron is equipped with tiny ion pumps that continually squeeze out positive sodium ions while sucking in negative potassium ions. The difference in ion concentration leads to a negative potential across the cell membrane of approximately 70 millivolts (resting potential). In locations where neurons connect to one another there are biological mechanisms called synaptic junctions (synapse). Neurons form synaptic junctions in a variety of ways. The three basic types involve a connection between an axon and a dendrite, between two axons, and between two dendrites [Snyder, 1985:30]. Figure A.2 is a schematic diagram of the typical synapse.

Referring to Figure A.2, the region above the presynaptic membrane contains vesicles filled with chemicals called neurotransmitters. Although there are many types of neurotransmitters, a

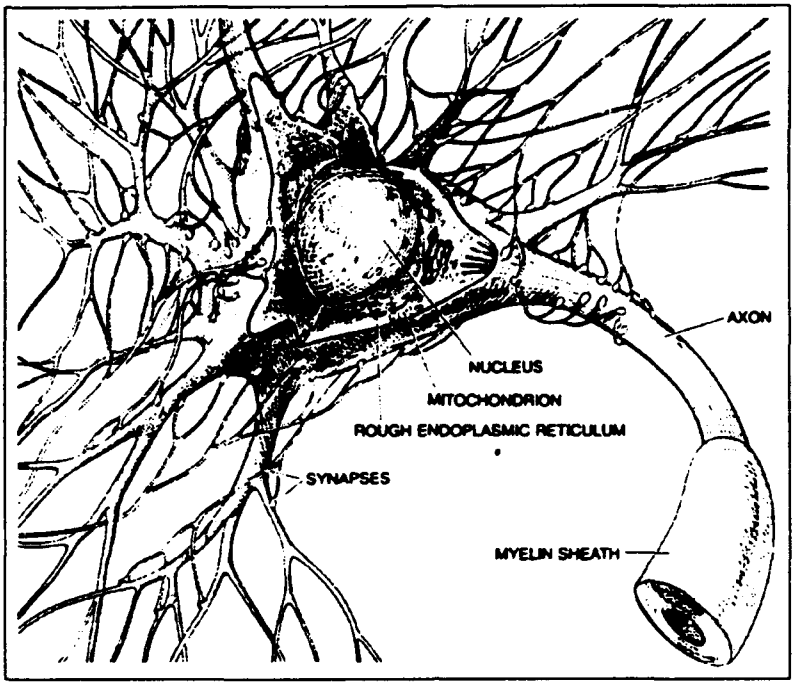

Figure A.1. Schematic of a Neuron [Stevens, 1979:55]

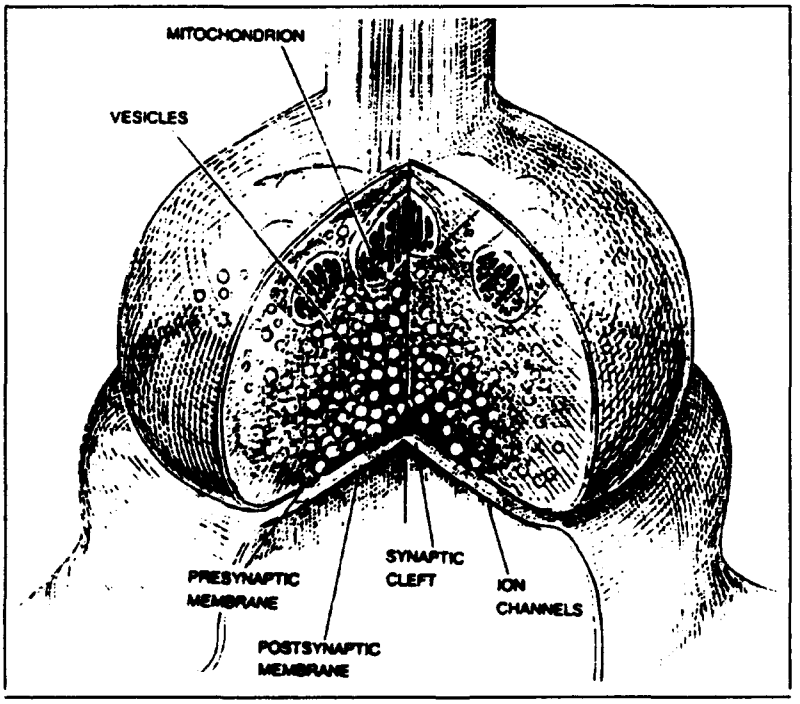

Figure A.2. The Typical Synapse [Stevens, 1979:56]

discussion of the various transmitter types is beyond the scope of this review. A synapse transmits information from one cell to another using transmitters in the following fashion.

First, a signal or voltage spike reaches the synapse and causes vesicles to fuse with the presynaptic membrane at the point where neural transmitters are released into the synaptic cleft. Next, the transmitters travel across the cleft to the postsynaptic membrane and activate ionic channels. The channels then allow ions to cross the cell's membrane and create a potential change in the intracellular fluid. If the change in the fluid is a positive excursion from the resting potential, the cell is depolarized and the interaction is called excitatory. On the other hand, if the change is a negative excursion, the cell is hyperpolarized and the interaction is inhibitory. When the cell is depolarized beyond a threshold value, the cell responds with a sharp rise in voltage called an action potential. The shape of the action potential (Figure A.3) depends on the previous state of the neuron.

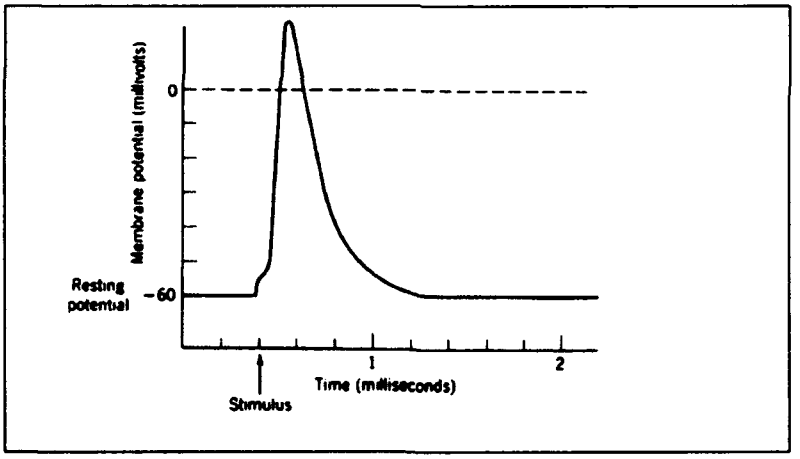

Figure A.3. A Cell's Response to Applied Stimuli [Taken from Levine, who adapted it from *C.F. Stevens, "Neurophysiology: A Primer," Wiley, New York, 1966, p.14*, page 89]

This section described a generic neuron that can be found throughout the body in different forms. The next section looks at how the neurons in the retina are different from the generic neuron.

A.2.2 Retinal Architecture. The human eye, as shown schematically in Figure A.4, is about 25 millimeters in diameter and is designed to gather light from a 208 degree arc centered about

the visual axis [Levine, 1985:68]. The bulk of the eye is designed to focus light onto a paper-thin

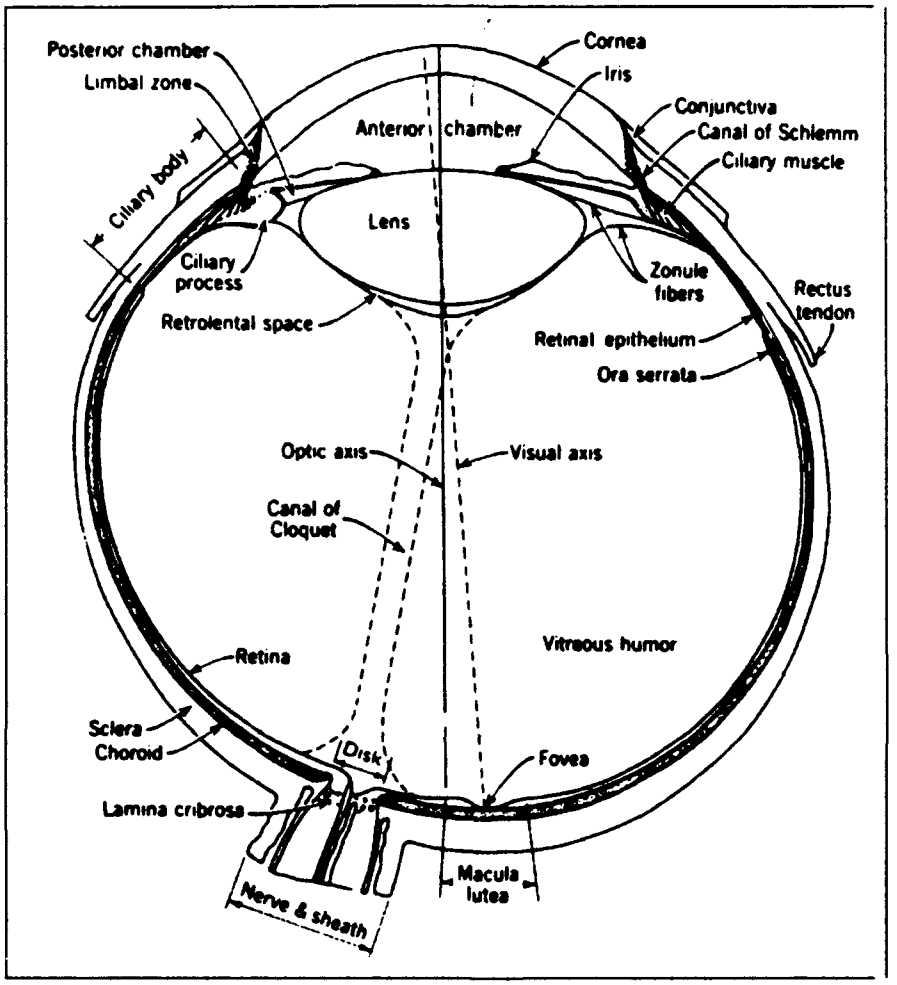

Figure A.4. The Human Eye [Taken from Levine who adapted it from a variety of other sources, page 61]

lining called the retina. The retina is responsible for transforming radiant energy in the optical frequencies into nerve impulses that our brain interprets as vision. From input to output, the retina compresses data from 126 million photoreceptors into signals that pass over some one million fibers located in the optic nerve [Levine, 1985:75]. Just what is the eye doing to compress such a large number of data points before transmission to the brain? Researchers have focused on this question for the last quarter-century and are just now beginning to model certain portions of the visual system.

A.2.2.1 Neural Architecture. Surprisingly, light must travel through the entire retinal tissue before reaching the detector elements of the eye. Surrounded by organic support tissue and covered in blood vessels, the retina spans 200 degrees of the rear wall of the human eyeball [Levine, 1985:72]. The structure of this sheet varies across its surface and thus provides regions of differing acuity. Surrounding the visual axis is an area of great acuity called the fovea centralis. This region covers an arc of approximately two degrees centered on the axis. The annular region outside the fovea centralis is the parafovea which subtends an angle of 8.6 degrees from the visual axis. The remaining area is commonly called the periphery or perifovea and accounts for approximately 97.25 percent of the retina [Levine, 1985:75]. Buried in the organic tissue lie five layers of neurons responsible for encoding the eye's imagery (Figure A.5). They are the photoreceptors, and the bipolar, horizontal, amacrine,[1] and ganglion cells (ganglia).

Once light strikes the layer of receptors the photons are changed from optical energy into electrical signals. The photoreceptors pass these signals to the bipolar cells and then to the ganglion cells where the signals are transformed into action potentials. The photoreceptor/bipolar/ganglion connection make-up the direct visual pathway [Masland, 1986:104]. Two other cell networks influence the signal as it moves along the visual pathway. The first is the horizontal network which lies below the photoreceptors and above the bipolar cells. The second, the amacrine cell network lies between the bipolar and ganglion cells. These two networks are termed the lateral visual pathways. After leaving the retina, the signal travels to the Lateral Geniculate Nucleus (LGN) where it is further processed and sent to the visual cortex of the brain. Except for the ganglion cells, retinal neurons are characterized by having only a soma and highly specialized dendritic trees. These cells have no axon because they need only transmit information short distances. The following sections describe each cell type in more detail.

[1] The proper pronunciation of amacrine is ä'mä-krĭn.

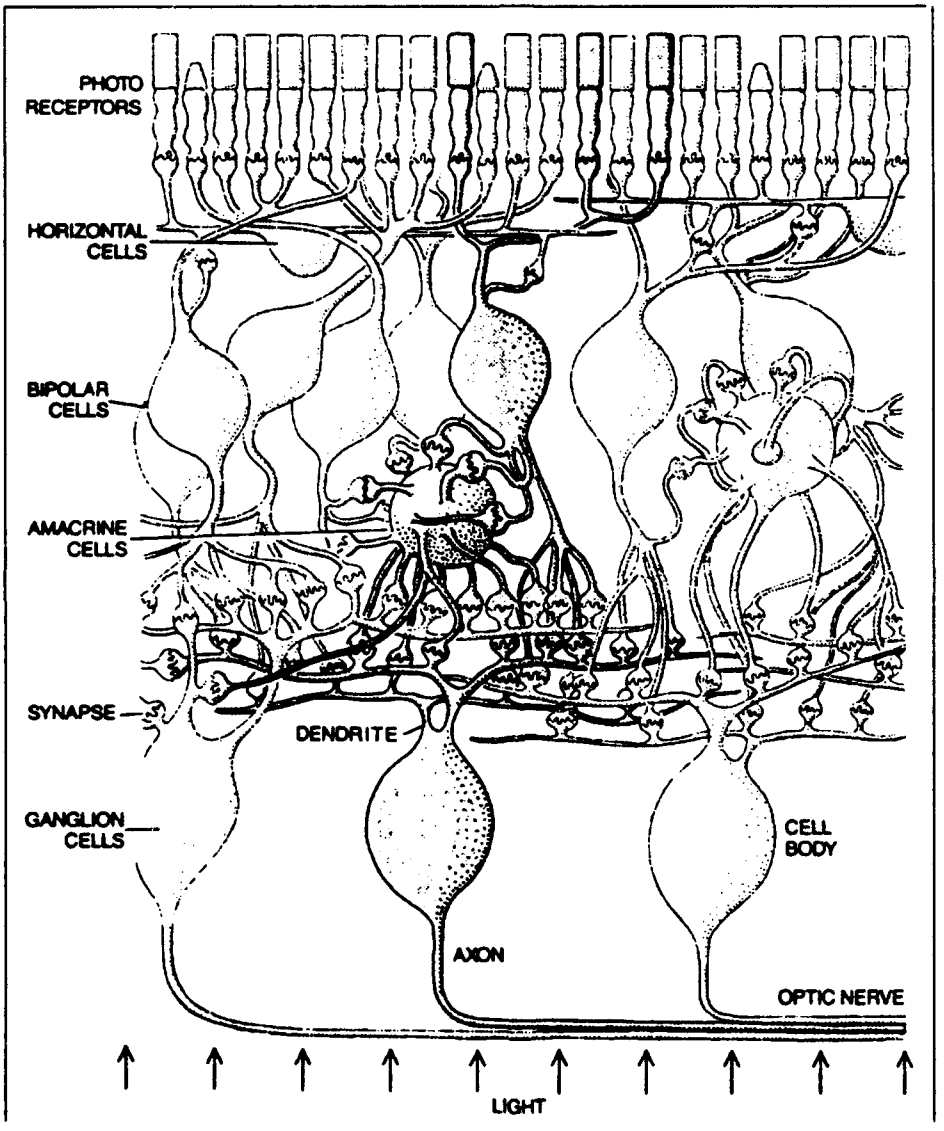

Figure A.5. The Retinal Architecture [Poggio, 1987:48]

A.2.2.2 Photoreceptors. Of all the specialized neurons in the human body, only the photoreceptors are capable of transducing electromagnetic energy (photons) into electric potentials. The two types of photoreceptors are named after their physical shapes (Figure A.6). Rod

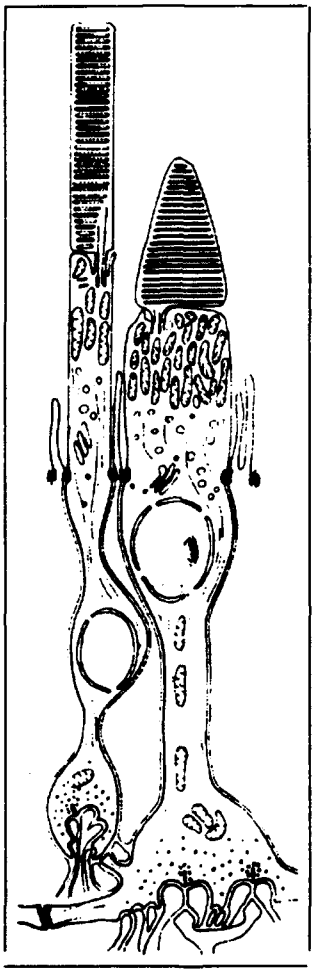

Figure A.6. The Human Rod (left) and Cones (right) [Taken from Levine from *L. Missotten, "The Ultrastructure of the Human Retina," Editions Arscia Uitgaven, N.V., Brussels, 1965* page 77]

shaped receptors are covered with a photopigment called rhodopsin. Rhodopsin responds to a relatively wide band of the visible electromagnetic spectrum (400 nanometers to 700 nanometers) [Levine, 1985:76]. Cone shaped receptors can be covered by one of three photopigments each covering a different, narrow band of the visual spectrum. Similar to the television camera, cone receptors provide color information centered in the three spectral bands [Levine, 1985:78]. Together, rods

and cones provide a visual sensitivity over ten logarithmic decades [Werblin, 1973:78] of photon intensity. The low end corresponds to dim situations encountered under star light and the high end under very bright conditions such as at high noon.

In low light conditions, the eye provides gray scale[2] information about the intensity of local imagery. This is accomplished by the rod cells which can integrate over the entire visual spectrum of photon energy. Not surprisingly, there comes a point where the total irradiance on the retina saturates the rod cells. Fortunately, by the time the total energy is capable of saturating the rod cells, cone receptors begin to respond to their varying spectral bands [Werblin, 1973:77]. They too integrate over their bandwidths but since the bands are relatively small, they do not saturate. Frank S. Werblin quantified the response of the two photoreceptors in 1973. Using a central spot of varying intensity surrounded by a background of constant intensity, Werblin plotted the response of rod and cone cells. In Figure A.7 Werblin illustrated that rod receptors (dashed lines) begin to respond at lower light intensities but saturate quickly (within three logarithmic decades). Cones are shown to respond at higher intensities but do not seem to saturate over the subsequent five decades following rod saturation. The combination of rod and cone response ranges allows the eye to maintain a constant sensitivity range about the level of background intensity.

As was pointed out earlier, the eye maintains different regions of acuity across its surface. This is accomplished by varying the density of rod and cone cells in each region. The fovea centralis is defined by the region where cones outnumber rods. In fact, in a 0.1 millimeter radius about the visual axis, the density of cones reaches 160,000 cells per square millimeter (cpm) and there are no rod cells [Levine, 1985:75]. In a 0.2 millimeter radial annulus, cone density decreases and rods increase until they equal cones at about 50,000 cpm each [Levine, 1985:75]. The density of rods continue to increase throughout the parafovea peaking out at 160,000 cpm while cone densities drop to 10,000 cpm. On the periphery, cone density remains constant and rod density falls off at

[2]The term gray scale is used to describe a series of monochromatic levels varying from white to black. An intermediate stage would be a GRAY level.

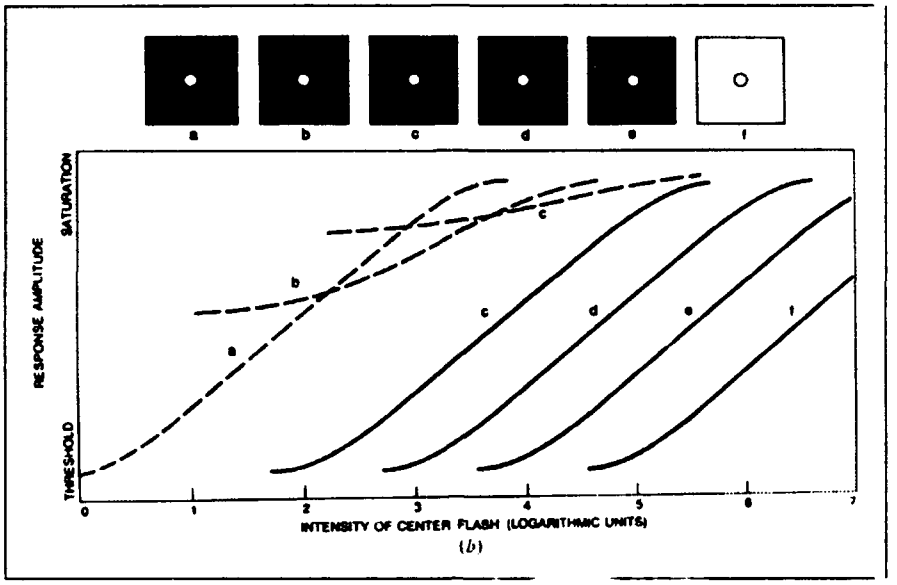

Figure A.7. Response of Rods (dashed) and Cones (solid) [Werblin, 1973:77]

a constant rate. The overall effect of receptor distribution is to provide more data on objects in the center of the field-of-view than those on the periphery. This data includes color and contrast information.

Directly below the receptor cells are the networks of bipolar and horizontal cells. The lateral, horizontal cells spread across many receptive cells while the bipolar neurons penetrate from the outer synaptic layer (location of horizontal cells) to the inner synaptic layer (location of amacrine cells).

A.2.2.3 Horizontal Cells. Werblin found that the response of the eye depends on spacial and temporal integration of a region called the receptive field. Levine defines the receptive field as *"a pattern of receptor cells in the retina which results in a specific firing behavior for a particular single cell anywhere in the visual pathway"* [Levine, 1985:33]. The receptive field is not a fixed anatomical structure and depends on several factors such as the objects in an image, the total intensity of the image, the retina's adaptation to the image, and the distance from the visual axis. The horizontal cells and the amacrine cells are thought to have a prime role in determining the

receptive field. Each receptive field has an inner excitatory region and an outer inhibitory region [Levine, 1985:103]. The regions are circular or slightly elliptical in form. Werblin termed the two opposing fields the center (excitatory) and surround (inhibitory) regions. Later we will see that there are two types of receptive fields – on-center, off-surround and off-center, on-surround.

There are two kinds of horizontal cells, large and small. It is thought that the horizontal cells provide an average spacial intensity which is used by the bipolar cell. The location where photoreceptor, horizontal, and bipolar cells meet is called the triad synapse [Mahowald, 1991:76]. The word triad implies that three cell types converge. However, more than one of each cell may contribute at the triad synapse. For example, the small as well as the large horizontal cell may both contribute to the function of the synapse. It may just be coincidence that there are two sizes of horizontal cells and there are two regions of the receptive field. Furthermore, it stands to reason that the smaller cell provides excitatory stimulus and the larger cell provide inhibitory stimulus to the bipolar cell.

One effect attributed to the receptive field is the occurrence of Mach bands. A mach band is the phenomena of perceiving a light band just inside the edge of a black bar which is adjacent to a white bar of low contrast. The opposite effect (a dark band) also occurs just inside the edge of the white bar. Figure A.8 illustrates this effect graphically. The gray scale image (a) has an intensity profile (b). However, if you stare at the image, you will perceive the profile to be (c). This effect is also known as edge enhancement. As will be discussed in more detail later, the receptive field is the summation of the intensity values of each receptor in the center and surround regions. The weights are a function of distance from the field center and have a gaussian profile. The center profile is a narrow positive curve and the surround is a wide negative curve. If the intensity is constant across the field, the summation of the two is a nominal value. Any discontinuity in the field causes an imbalance of the two regional values and the summation will deviate from the norm producing Mach bands. The cell summing the center-surround regions is the bipolar cell.

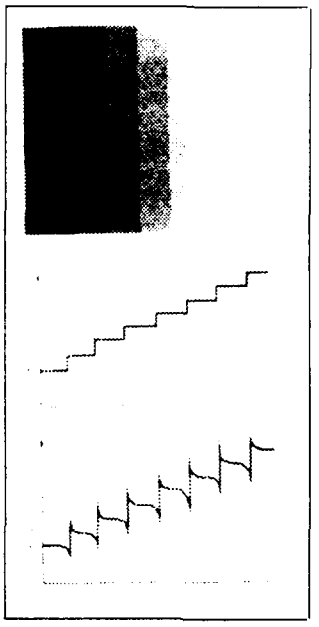

Figure A.8. Mach Band Phenomena [Lavine, 1985:222]

A.2.2.4 Bipolar Cells. The bipolar cell by all accounts is a pathway for the combined outputs of the horizontal cell and photoreceptor. There are four types of bipolar cells. They are the rod bipolar, the invaginating midget bipolar, flat midget bipolar cell, and the flat bipolar [Levine, 1985:81]. The rod bipolar cell connects only to rod receptors while the other three types connect to cone receptors. In the fovea centralis, each cone projects to a single bipolar neuron [Levine, 1985:81]. Outside this region, the number of photoreceptors projecting to a single bipolar increases. At 20 degrees from the visual axis, hundreds of cones connect with one bipolar [Levine, 1985:81].

In contrast to the notion that the two types of horizontal cells provide the center (spot) and surround regions of the receptive field, the multiple connections of the bipolar cell might alternately provide this type of field. In the center of the fovea, the spot may only be a single photoreceptor wide with the horizontal cells providing the surround inhibition region. As we move outside the fovea centralis, the receptive field grows linearly with distance from the visual axis [Grzywasc, 1990:45]. With increased size, the contribution of a given photoreceptor to the center-

surround regions depend on the relative distance from the central point of the field. As we will see later, the center and surround region's contribution can be modeled using two gaussian curves with different spreads (Figure A.10).

Werblin characterized the response of the bipolar cell in the same way as he did the photoreceptors. Figure A.9 illustrates how the bipolar cells retain a steep response curve that shifts from the lowest response level of the rod receptor to the highest level of the cone receptor. At this point,

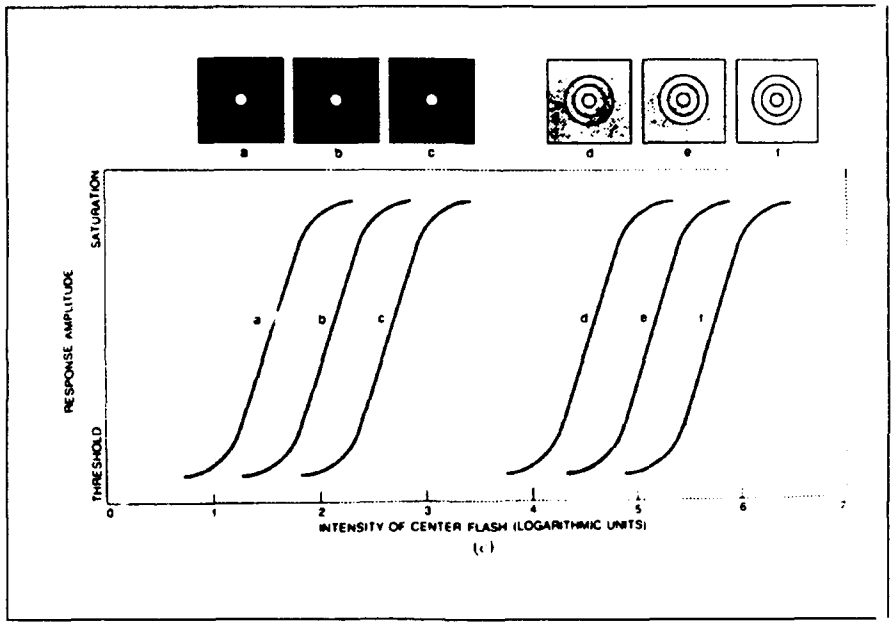

Figure A.9. Response of Bipolar Cells [Werblin, 1973:77]

the retina has *discounted the illuminance* in the eye's field-of-view. In other words, the imagery received by the photoreceptors is normalized to a narrow band of intensity levels and the ambient lighting conditions of the scene are adjusted to a standard scale. Additionally, the steepness of the bipolar response curve is characteristic of high contrast photographic film [Werblin, 1973, 76]. Werblin concluded that the contrast of the scene was enhanced due to the narrow bipolar response curve.

Having passed through the top three layers of the retina, the eye has adjusted the visual image for ambient lighting conditions and enhanced both the contrast and edges in the scene. In

the next two layers, the image is further processed to detect motion before transmission to the LGN along the optic nerve.

A.2.2.5 Amacrine Cells. As mentioned earlier, the amacrine cells provide a second lateral pathway for signal propagation as they regulate the behavior of the ganglia. When light strikes the retina, the amacrine cell responds immediately but with continued light, the response dies out. This type of response is thought to sharpen the transient responses of the ganglia. The amacrine cell is thought to have as many as 30 morphologically different forms [Masland, 1986:106]. The four amacrine neurons discussed in this review are the cholinergic, the AII, the dopaminergic, and the indoleamine-accumulating amacrine cells. These cells are named after the neurotransmitters they use.

The *cholinergic* cell is most abundant in the periphery of the retina. Here, each bipolar cell is covered by as many as 140 cholinergic cells [Masland, 1986:108]. It is believed that unlike other cell types, the cholinergic dendritic tree are electrically isolated from one another. This gives them the ability to respond to spots smaller than the dendritic spread of the cell [Masland, 1986:108]. Conversely the AII cell has a much smaller spread of dendrites than does the cholinergic cell.

The *AII* cells cover the entire retina but overlap very little. They respond as do other amacrine cells but are thought to provide a spacial summation of bipolar rod cells which is useful in dim light situations. In fact, the only direct path for rod receptors is through the AII cells [Masland, 1986:109]. In low light, the AII cells form a direct pathway for rod cells by summing the intensity of the rod bipolar cells and causing a ganglion cell which otherwise would not fire. Both the cholinergic and AII cells have densely packed dendritic tree structures, a quality the dopaminergic cell does not have.

In the *dopaminergic* cell the dendritic tree is a spotty mosaic of branches with a wide lateral spread. This cell does not have synapses with ganglion cells [Masland, 1986:110]; rather, it connects to other amacrine cells. Little else is known about the dopaminergic cells. It is thought that this cell is involved in some other function than spacial resolution. The last cell type I will mention is the indoleamine-accumulating cell of which there are five distinct types.

The *indoleamine-accumulating* amacrine cell forms a dense plexus or network in the innermost margin of the inner synaptic layer and connects with most of the rod bipolar cells. As a whole, this family is thought to form five different pathways by which other retinal cells can interact. Since they are associated with rod photoreceptors, these pathways may provide a variety of subfunctions that are required under low light conditions [Masland, 1985:110]. Only the amacrine cells synthesize and secrete the excitatory neurotransmitter acetylcholine (ACh) [Poggio, 1987:51], a chemical that has been shown to provide directional information in the retina of turtles. Directional selectivity is a function of the final cell type, the ganglion cell [Poggio, 1987:51].

A.2.2.6 Ganglion Cells. Having passed through a variety of neural cells, image information detected on the photoreceptors is transformed from graded potentials to action potentials and sent along the optic nerve to the brain. The final layer in the visual pathway is the ganglion cell. It is the only cell capable of producing an action potential. The output fibers of the ganglia form the optic nerve. To perform this function, the ganglion cells manifest themselves into three types - on-center, off-surround units, off-center, on-surround units, and on-off units [Levine, 1985:102]. As indicated by their names, each type responds differently to the detected stimuli. The On-unit responds with action potentials to the introduction of light stimuli and makes up 20 percent of the ganglia. The action potential dampens exponentially with time to the stimuli and eventually adapts to the signal. Off-units respond similarly to the removal of a light source and make up 30 percent of the ganglia. The on-off ganglia comprise the last 50 percent and respond to a change in light. Ganglion cells also respond to spacial distribution of light intensity.

R. W. Rodieck derived an analytical model for the receptive field cell mapping. He assumed the ganglion cell had a temporal and spacial response that were separable. He postulated a function, $f(x,y,t)$, which was the product of the spacial and temporal functions, $W(x,y)$ and $h(t)$, respectively. The form of $W(x,y)$ is a three dimensional figure that resembles a mexican sombrero comprised of two gaussian functions of differing variance. The spacial function modeled the excitatory and inhibitory aspects of the receptive field well. Both the spacial and temporal functions are shown in Figure A.10.

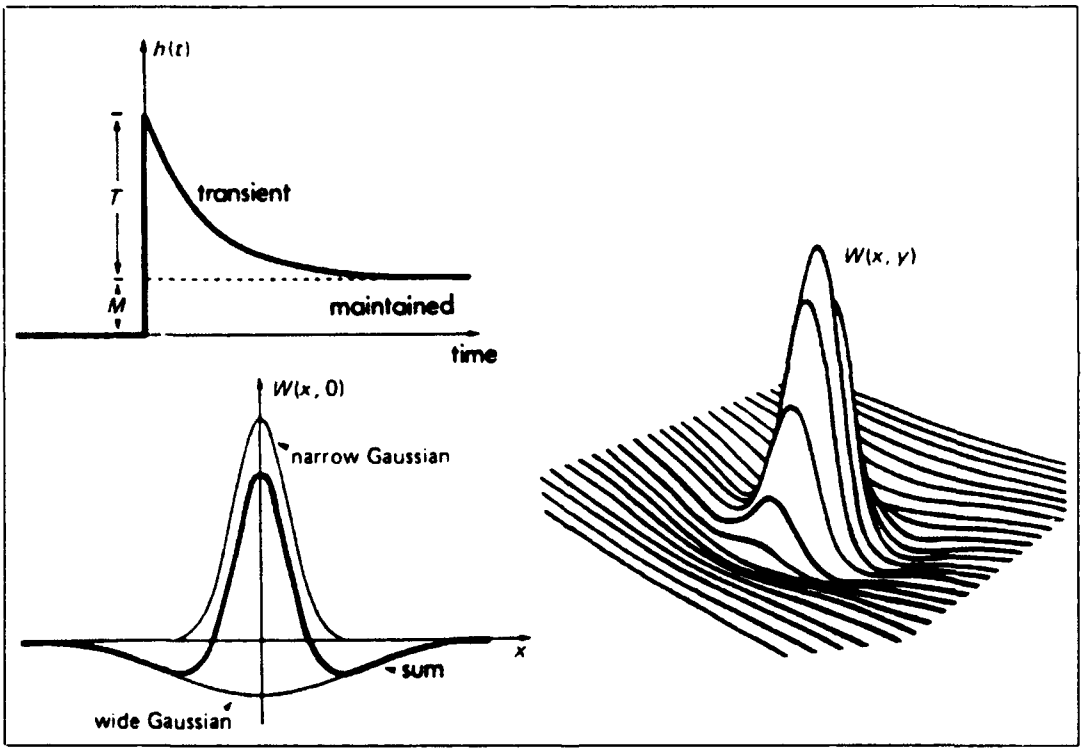

Figure A.10. The Functions $h(t)$ and $W(x,y)$ [Taken from Levine, from *R.W. Rodieck, "Quantitative Analysis of Cat Retina Ganglion Cell Response to Visual Stimuli," Vision Research, vol. 5, no. 11/12, 1965, pp. 583-601*, page 107]

In 1987, Tomaso Poggio and Christof Koch reported in Scientific American that the eye uses ganglia that are directionally sensitive. That is, they output only when an object is moving in a preferred direction. Poggio and Koch noted that a veto system was in use that depends on the excitation and inhibition signals having different propagation times [Poggio, 1987]. If the image of

an object strikes a pattern of receptors creating inhibitory and excitatory signals simultaneously, the faster excitatory signals will propagate to the ganglia more quickly than the inhibitory signals causing a response in the optic nerve. If the image is moving in a *preferred direction*, excitatory signals will be created before inhibitory signals and enhance this effect. However, in the opposite or *null direction*, the inhibitory signals are created first and are able to reach the ganglia before or simultaneously with the excitatory signals and are able to cancel their effect. Thus the ganglia does not respond in the null direction. The key to directional selectivity is to keep the inhibitory signal from depolarizing the ganglion when no excitatory signal is present. That is, the inhibitive signal causes excitatory signals to be *shunted* to the extracellular fluid but does not cause a change in the postsynaptic neuron. Therefore, the synapse must use *Shunting* or *Silent* inhibition.

A.2.3 Summation. The retina is composed of millions of specialized neurons. In general, each neuron responds to the inputs from a host of neighboring nerve cells by way of action potentials. Once the action potential reaches the synapse, it causes vesicles to inject neurotransmitters into the synaptic cleft. The transmitters bind with ion pumps imbedded in the postsynaptic membrane and cause the pumps to flow ions into or out-of the target cell. This change in potential is sensed by the neuron's soma and the cycle continues.

In the retina, five classes of neurons are arranged in a dense plexus of interconnections to process visual information. The photoreceptors, horizontal cells, and bipolar cells form the outer layers. The outer layers transform light imagery and normalize it to a standard range for use in the inner layers (amacrine and ganglion cells). The scene has passed through the first receptive field and now has enhanced edges and contrast in a narrow band of intensity values. The inner layers use receptive fields formed by a variety of amacrine cell types to enhance the transient response of ganglia. Three types of ganglia have been identified: On-center, Off-center, and ON-OFF units. The ganglion cells respond to motion in a directionally selective way. Each cell has a null and

preferred direction that is the result of a specialized inhibitory mechanism called Shunt inhibition. The next section reviews some mathematical models used to simulate the retina.

A.3 Emulating the Retina

A.3.1 A Scale Invariant Strategy for the Retina. In the scale invariant sampling strategy, a lattice of nodes is superimposed over the retinal surface. Each node represents a sample location for incident radiant energy (light) or sampling node. The nodes are not constantly spaced but vary linearly with eccentricity from the fovea [Grzywasc, 1990:45]. Figure A.11 illustrates the sampling

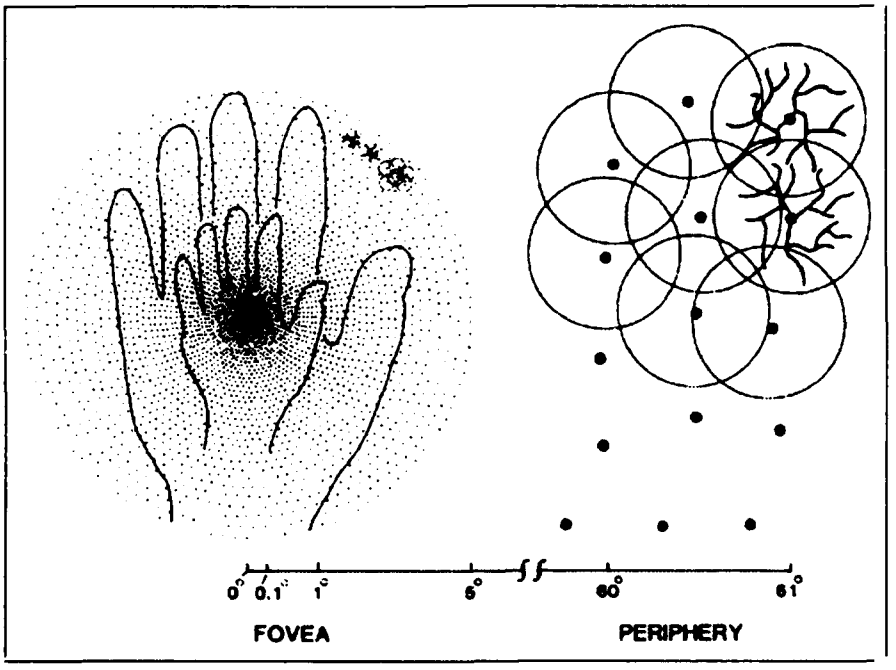

Figure A.11. A Scale Invariant Sampling Strategy for the Retina [Grzywasc, 1990:46]

lattice. Just what is a scale invariant strategy and how the sampling lattice is used to support it first requires the definition of two terms: the sampling interval and sampling area. The sampling interval is the distance between two adjacent nodes. This interval increases linearly with distance from the visual axis. As each node integrates information provided by neighboring nodes, the area over which the node integrates is the sampling area. Contribution from each node in the sampling area is weighted according to its distance from the integrating node. The weight profile is assumed

to be gaussian [Grzywasc, 1990:45]. Both the sampling area and interval are scaled proportionately to create similar overlaps between adjacent nodes across the entire lattice. This strategy leads to the property of scale invariance. Scale invariance provides equal data about objects regardless of the size of such objects. As an example, both the large and small hands in Figure A.11 are sampled by an equal number of nodes.

Mathematically speaking, a completely scale invariant system would require the sampling interval to decrease to zero at its center. However, real photoreceptors have finite size and can only reach a minimum value. The minimum interval in the retina is an estimated 0.01 degrees [Grzywasc, 1990:47]. An equation describing the expansion of sampling intervals (D) based on the eye is given in equation A.3.1 [Grzywasc, 1990:47].

$$D = \delta + \alpha E \tag{A.1}$$

Here δ is the minimum interval (0.01 degrees), α is the rate of interval increase with eccentricity E. The rate is estimated at 0.01 degrees per degree eccentricity. What this all means is that the spacing from the fovea centralis (0 degrees) to the far periphery (60 degrees) changes 50-fold and the area increases 200-fold [Grzywasc, 1990:47].

Ultimately, the entire visual field can be sampled using only 300,000 sampling nodes. This represents the most significant reduction in data for the retina as it discards 99 percent of the visual information received by the photoreceptors. In contrast to this strategy, one that samples the entire visual field uniformly at 0.01 degrees would require hundreds of times as many samples [Grzywasc, 1990:47].

The nodes provide information about a small region of the visual field. Theoretically each node could be mediated by a single optic fiber however, additional information and fault tolerant can be provided by a system that allocates multiple fibers per node. The retina seems to employ

this strategy [Grzywasc, 1990:47]. Ganglia are known to operate at two spacial scales leading to the idea that the retina uses two distinct sampling lattices.

A.3.2 "The Silicon Retina" [Mahowald, 1991]. One of the most successful attempts to create a machine solely to model early vision was done at the California Institute of Technology (Caltech) by Carver Mead. Mead and Mahowald produced a microchip that emulates the top three cell layers of the retina; the photoreceptors, horizontal cells, and bipolar cells.

The Silicon Retina demonstrates that adaptation is a key function in the retina. Adaptation as hinted to earlier, is the ability of a neuron to 'get used to' stimuli. In order for the eye to respond to varying conditions across the visual field, it needs to locally adjust its sensitivity to brightness. That is why we can see objects in bright conditions and shadows simultaneously. Adaptation, as modeled by the Silicon Retina, is accomplished using the lateral connections of the horizontal elements.

The horizontal connections provide a spacial average of the photoreceptor outputs. This, in essence, provides a smooth intensity surface that the bipolar cell uses as a reference to evaluate the output of the receptor element. Both the horizontal and receptor elements produce logarithmic signals which when subtracted by the bipolar element yields a ratio of local light intensity to background intensity. Thus, the interaction has discounted the illuminance[3] in the scene.

Another consequence of adaptation is that large, uniform regions closely match the smoothed intensity surface. The bipolar element interprets this as a difference of zero or an intensity ratio of one. However, edges which represent a large contrast are interpreted as large differences by the bipolar element and they stand out as ratios greater than or less than one. Now consider a surface represented by the output of the bipolar elements. Locations with high contrast will stand out

[3]The illuminance of a scene is a measure of how it is lighted. The only important aspect of the scene is the reflectivity of various objects.

as deviations from the smooth surface. In this manner, important information about the scene is obtained almost immediately.

The temporal response of the horizontal elements are slower to decay than that of the photoreceptors. When an object moves across the spacial field of the receptors, the difference between the current intensity profile and the temporally lagging smoothed surface is intensified until the surface can catch-up to the motion.

The Silicon Retina models the receptor element by using a bipolar transistor[4] with a feedback loop. The transistor *"produces a current proportional to the number of photons it absorbs"* [Mahowald, 1991:79] and the feedback loop *"amplifies the difference between the instantaneous photocurrent and its long-term average level"* [Mahowald, 1991:79]. In this way, the receptors mimic the time response of a real cone cell. The sensitivity of this element is 100,000 photons per second (moonlit conditions). Due to the feedback loop, large changes in intensity will temporarily saturate the receptor, just as if you looked into a bright light from a shadow.

The horizontal elements are a network of resistors and capacitors. The resistors form a hexagon with a central node. All seven nodes of the figure are connected with identical resistive values (Figure A.12). The capacitors of the network model the time dependence of the horizontal cells. Each node of the network represents a weighted average of the receptor outputs and the value of the resistors determines the area over which the smooth curve integrates.

The bipolar elements are amplifiers that sense the voltage difference between the network and receptors. One amplifier is used for every node of the network. Additional circuitry follows the bipolar element to allow researchers to study various parts of the circuit. A revision of the receptor segment of this model was given in 1991 by Misha Mahowald. The result was a more reliable model [Mahowald, 1991].

[4] Bipolar Transistors are in no way related to the Bipolar cells.

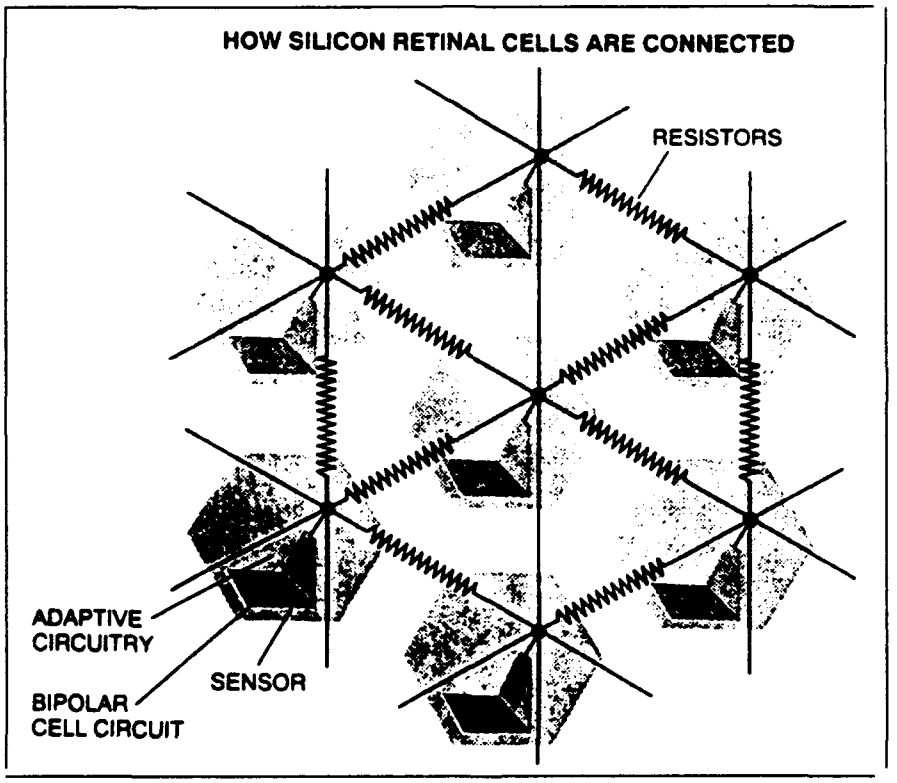

Figure A.12. The Hexagonal Resistive Network of the Silicon Retina [Mahowald, 1991:78]

Test results from this model have paralleled biological data remarkably well [Mead, 1988:94]. Correlation with the results obtained by Werblin and those indicated by Stevens have won acceptance of this model. So much so that the Air Force has contracted Amber Engineering to produce a number of chips based on the Silicon Retina.

A.3.3 Infrared Focal Plane Detectors (IFP).

A.3.3.1 "The Neuromorphic Infrared Focal Plane" [Massie, 1992]. The Neuromorphic Infrared Focal Plane (NIFP) is a microchip based on the Silicon Retina model and has been nick named "Smart Retina" by its designers at Amber Engineering. A focal plane array is an array of elements like the receptors described in the Silicon Retina section that are located at the focus of some optical system. The purpose of the Smart Retina is to preprocess infrared data in hopes of reducing the post- processing required by digital computers. The NIFP is intended to be a missile seeker device for use in the Strategic Defense Initiative (SDI) program.

The NIFP has been designed so that the operators can tune the focal plane's spacial response via off-plane control of spacial and temporal filters. The NIFP has a 64 x 64 indium antimode focal plane array. The actual device is not yet available but the company has developed simulations showing the proposed performance to closely parallel Mead's Silicon Retina. Operational devices are projected to be available by the end of this summer (1992).

A.4 Conclusion

This review introduced the neuron, the human retina, and neural vision models. It showed that the general neuron's activity is dependent on synaptic junctions which in turn use a variety of neurotransmitters to convey information across the synaptic cleft. Action potentials are key to the operation of the neuron and are instrumental in understanding the process of adaptation. The human retina has five specialized neurons arranged hierarchically over the back of the human eye.

Photoreceptors transform light into electrical signals that propagate through bipolar and ganglia in the direct visual pathway. Lateral interactions are provided in two layers of the retina.

The layer of horizontal cells furnish lateral connections required in forming receptive fields. The two types of receptive fields are characterized by on-center units and off-center units. Amacrine cells also form lateral connections and contribute to a secondary receptive field and motion sensitivity. Experimental data show as many as 30 different types of amacrine cells covering a wide range of dendritic tree density and lateral spread. The amacrine cells depend on lateral inhibition as well as silent inhibition to provide directional selectivity in the ganglion layer.

The ganglia form the last layer of the retina and produce action potentials along the optic nerve. In addition to the two basic types of receptive field units, the ganglia have a third type; on-off units. These units represent the motion sensitive portion of the retina's output. Dependent on the amacrine cells, the ganglia react to motion in preferred directions only.

In an effort to learn from the neural systems, researchers develop hypotheses to explain experimental data. The resulting models are tested against observed data by the use of models. The sampling lattice model places sampling nodes over the retina. The spacing is linearly dependent on the distance from the visual axis. Each node has a sampling area that increase 200-fold from center to periphery. The effect of this lattice is to provide scale invariance to objects projected onto the retina. Scale invariance provides a reduction of 99 percent of the image data gathered by the photoreceptors.

In an attempt to emulate the retina, Carver Mead and Misha Mahowald developed a microchip that emulates the top three layers of the retina. The chip discounts the illuminance in the visual field, enhances edges, and sharpens the contrast. The Neuromorphic Infrared Focal Plane, based on the Silicon Retina design, provides a capacity to tune the focal plane's spacial response and is intended for missile seeking applicatio. s.

This review identified trends in retinal modeling. Much like the evolutionary process that created the retina, researchers must make their design steps incremental. The Silicon Retina was the first successful step.

Appendix B. The Mechanism of Performing a Simulation

This appendix is designed to give the reader an idea of what is involved in performing a simulation of our models. We discuss the steps taken to prepare real input imagery as well as how the model is configured to accept the images. We give the source code for the incidental software used in preparation and post processing the images. The source code for the model software is given in the Appendix C.

B.1 Image Preparation

Figure B.1 shows the step taken to get our images into a form accepted by our simulation software. The images are obtained by capturing them using a virtual image processor and is

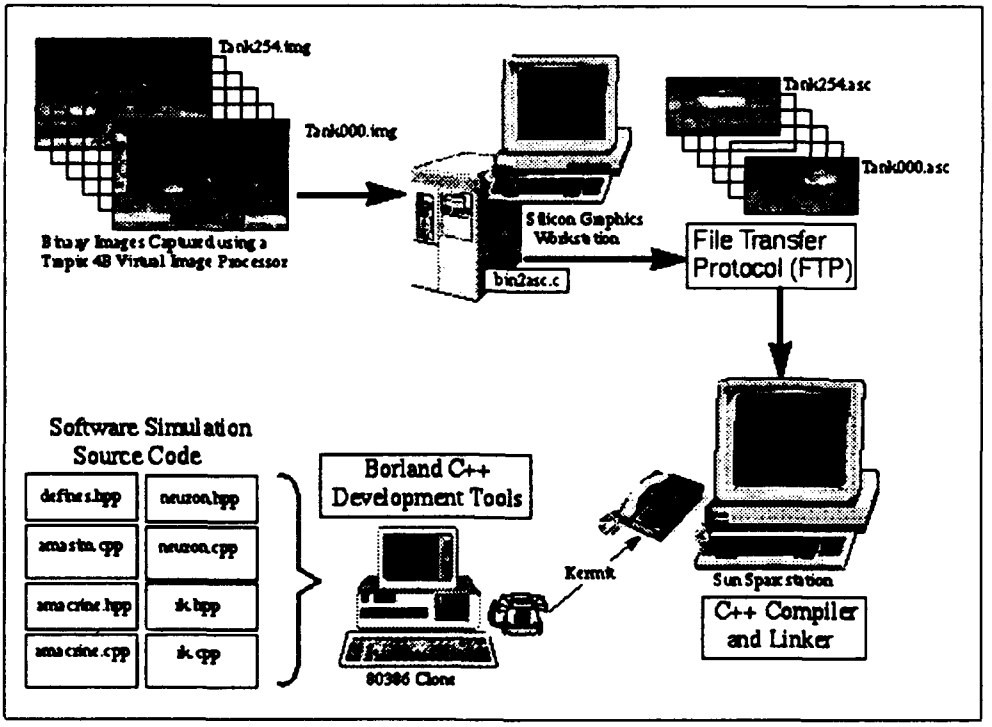

Figure B.1. This diagram shows the steps taken to prepared the model and images for simulation. The process involves several computer systems and source files.

explained in Appendix D. We used a variety of computers for several reasons. The Silicon Graphics computers were initially used for their speed. However, due to the low number of Silicon Graphic

machines available, we opted to perform the major modeling on the Sun Spacstations. This allowed us to tuse the storage capacity on the Silicon Graphics workstations and distribute the workload. The images were coded in a binary format and had to be transfered to ASC II representations. We coded a program called `bin2asc.c` which allowed us to identify which pixels to save after the conversion. The source code for this program is given below. Once the files were created, they were transfered using the unix `ftp` command to the Sun workstaions.

B.1.1 Source Code: `bin2asc.c`

```
#include <stdlib.h>
#include <stdio.h>
  // The following line is used to identify whether the program
  // is to run on the NeXt machines or the Silicon Graphics computers.
  // This definition is required because the two have different integer
  // sizes as defaults.  The NeXt uses a two byte integer while the
  // Silicon Graphics uses a four byte integer.  However, my data is
  // in binary and will not correctly be loaded if this is not set
  // Right.  If you use this on a Silicon Graphics computer, change
  // "NeXt" to "SGI".
#define SGI
#ifdef SGI
#else           define INTSIZE short int
#endif          define INTSIZE int

main(argc, argv)
int argc;
char *argv[];
{
     FILE *infile, *outfile;

     // When the data is read in, it can be place directly into
     // place holders of the correct length.  Here I use a structure
     // to parse the data in to discardable and retainable data. When
     // the structure is filled using two bytes, the high order bits
     // will fill the 'sixbit' bitfield and the low order bits fit
     // into the 'eightbit' bitfield.  Finally we get to my twobits
     // which isn't worth keeping so it too is discarded.
     struct bitfield {
        unsigned INTSIZE sixbits   : 6;
        unsigned INTSIZE eightbits : 8;
        unsigned INTSIZE twobits   : 2;
     };

     // Here I define a buffer to hold one row of the image in
     // an array of bitfields 1024 elements long.
     struct bitfield pixel[1024];

     // Throw in a few counters for fun.
     int i,j;

     if(argc != 3) {
        printf("\n Command error\n\n\t>bin2asc infile outfile\n\n");
        return 0;
     }

     // First I open the binary file as a read only file.  If this goes
     // smoothly, no errors, continue by opening the output ascii file.
     if ((infile = fopen(argv[1], "r")) != NULL) {
        outfile = fopen(argv[2], "w");

     // Next I gather each row of 2048 elements. */
     for (i=0;i<512;i++){
         fread(pixel,sizeof(pixel[0]),1024,infile);
```

```
            // Since the images don't fill the entire area, I clip
            // the lower 67 lines.
            if (i>146 && i<275) {
               for (j=69;j<325;j++)
                  fprintf(outfile,"%d ",pixel[j].eightbits);
                  fprintf(outfile,"\n");
               }
            }
            // Time to close the files
            fclose(infile);
            fclose(outfile);
         }
         else {
             return 0;
         }
}
```

B.2 Simulation Software Configuration

The source code was designed on a personal computer and transfered over the phone lines to the Sun workstations. Written in object orient C, the code had to be compiled using a C^{++} compiler. All the computers at AFIT have this capability but only the Sun workstations was easily accessable and had the hard-drive space to store the output. The `defines.hpp` header files contains the preprocessor statements that configure the model software. This file is included at the end of this section.

The software can get to be rather large. When we ran it on an 80386 machine, it was configured for a 64 by 64 pixel FOV. The executable file was some 177 kilobytes large and required 400 kilobytes of RAM with the dynamic linked list of past synapse values gobbling the remaining 140 kilobytes of RAM. We used DOS overlays which are available only in Borland C++. Without them, we may never have gotten a FOV of more than 45 by 45. The time to run our simulation was on the order of ten hours for a 64 frame input. When we ported the software to a Sun workstation, the 64 by 64 model took more hard drive space but since the Suns use a virtual memory system, the amount of RAM required was not important. One the Suns, we were able to run a 256 by 128 model in less than two hours (256 frames) but the model files size was well over 3 megabytes. The model output files were nearly 300 kilobytes and so disk space was always a problem.

B.2.1 Configuration Header File: `defines.hpp`

```
// Both ROWS and COLS must be multiples of 2.
#define ROWS   128 // This must be 2*FrSIZE+1 (at least)
#define FrSIZE 2   // Field (inhibitory) row size
#define ErSIZE 1   // Excitatory row size
#define COLS   256 // This must be 2*FcSIZE+1 (at least)
#define FcSIZE 4   // Field (inhibitory) column size
#define EcSIZE 2   // Excitatory column size
// #define LINEAR 0  // This is a switch to allow the first two lines of the
                     // Run command file to be used as output files
```

```
#define ASYNC 0
#define DELAY_TIME 0.05    // This is the delay in terms of the Nyquist Rate

#define TOLSYNAPSE 0.0005
#define SYN_ALPHA  0.5
#define SYN_BETA   5.0

#define TOLSOMA            0.0005
#define SOMA_DECAY         5.0
#define SOMA_UPPER_BOUND   45.0
#define SOMA_LOWER_BOUND   45.0
#define PI 3.14159265

// #define ECHO
// #define Ogmen 0     // remove the comment indicators for the O&G model
// #define GNUPLOT 0

enum FLAG {YES, NO};
```

B.3 Simulation Control

Early in the development of our software we decided to build an interpreter into our *main()* routine. The interpreter allows us to use a single executable program to respond to a variety of model inputs. We designed the software to accept a command line argument indicating the name of the command file. The command file controls the operations of the model by providing input image information as well as output filenames. As an example, suppose we built a command file called `sim.run` (the contents of which will be described later). To run the model compiled and linked to an executable file named `async.exe` we need only type;

```
>async sim.run
```

The format of the `sim.run` file depends on the needs of the simulation. In command files for one-dimensional models, the first two lines identify image filenames into which a time history of the model output is placed after every timestep as specified by the user (we will see how this is done later). Images of the synapse array and input values are stored in the first file while the second file is used to store images of the soma outputs. Command files used for two-dimensional models do not use these lines. Models that are defined in two dimensions use a different method of output storage as described below. The third line of the command file must be an initialize line.

Following the initialize line, the user controls the model by identifying input and output filenames with either of two command-file line formats. There is no limit to the number of lines following the initialize line. The simulation will halt after the last line in the file or when an ill-formatted line is encountered. The initialize line and command lines are discussed below followed by an example command file.

- **Initialize** lines are prefaced by `-i` or `-I`. Following the line preface is a `filename` which contains ASCII values representing the input to each node of the model. The inputs are applied in a row by column fashion starting with row 0, column 0. The input values must

be separated by spaces or carriage returns. The initialize command tells the model to apply the inputs and force the synapse and soma objects to steady-state. The first history cell is created at this time in the asynchronous models. An example of the initialize line might look like: `-i data/nrn_0.dat`

- **Advance** lines instruct the model to use some input pattern and propagate the model over an interval of time. The operation can be repeated without a change in input pattern. This line begins with either `-a` or `-A` and is followed by a `timestep` (real), the number of `iterations` (integer), and an input `filename`. The input file is identical to the input file discussed in the initialize line. After each advance of `timestep`, the program writes images to the storage files given in lines one and two of this file. A subtlety exists in this command line. If one were to propagate the model over a period of one minute, one could choose a `timestep` of one minute and an `iteration` of one. In this case, the image files will contain a single model image created one minute into the simulation. If one chose to use a `timestep` of one second and an `iteration` of 60, then the image files would have 60 images, created every second of the simulation. An example of the advance line might look like: `-a .05 10 data/nrn_1.dat`
- **Snapshot** lines are used when the model is defined in two spacial dimensions. This line begins with either `-s` or `-S` and is followed by a soma image `filename` and a synapse image `filename`. Since the image files used to record time histories of the model can only handle three dimensions (node column, time, and node value), a two-dimensional model requires some other method of storing the response of the simulation. Thus, we use the snapshot line to tell the model when to store an image of the soma and synapse arrays as well as where to store them. The four items saved in this manner are the node's row and column location, its value, and the time of the storage. Snapshot files are formatted identically to the input file formats already described. An example of the snapshot line might look like: `-s data/nrn_1.inp data/nrn_1.dat`

Examples of the command files follow. The first example is for a one-dimensional model.

```
a:/nrn.inp
a:/nrn.dat
-i data/nrn_0.dat
-a .05 10 data/nrn_1.dat
-a .05 10 data/nrn_2.dat
-a .05 10 data/nrn_3.dat
-a .05 10 data/nrn_4.dat
-a .05 10 data/nrn_5.dat
-a .05 10 data/nrn_6.dat
-a .05 10 data/nrn_7.dat
-a .05 10 data/nrn_8.dat
-a .05 10 data/nrn_9.dat
```

This file tells the simulation program to save images in the files `nrn.inp` and `nrn.dat` on the `a:` drive. The first input is saved in the file `data/nrn_0.dat`. Thereafter, the model is advanced every 0.05 time units. Each line has an iteration count of 10 and uses an input file of the form `data/nrn_#.dat`. These files represent an input pattern that is shifted one node location in each consecutive file. Therefore, by changing the input filename, this command file presents the illusion of motion. Input patterns are changed every 0.5 time units. The speed of the image can then be measured as 2.0 nodes per time unit (N/TU). If `interval` was set to 5, the image would seem to move faster - once every 0.25 time units or 4.0 N/TU. Speed is simply the reciprocal of `timestep` times `iterations`. Remember that the model is sampled every `timestep`, so this command file samples the model once every 0.05 time units. The sample rate is 20 samples per time unit (S/TU). These measures of image motion will be used to describe the model performance in the sections to follow.

A second example of the command file format is given below. This file is used for a two-dimensional model.

```
-i data/nrn_0.dat
-s data/nrn_0.inp data/nrn_0.dat
-a .05 10 data/nrn_1.dat
-s data/nrn_1.inp data/nrn_1.dat
-a .05 10 data/nrn_2.dat
-s data/nrn_2.inp data/nrn_2.dat
-a .05 10 data/nrn_3.dat
-s data/nrn_3.inp data/nrn_3.dat
-a .05 10 data/nrn_4.dat
-s data/nrn_4.inp data/nrn_4.dat
-a .05 10 data/nrn_5.dat
-s data/nrn_5.inp data/nrn_5.dat
-a .05 10 data/nrn_6.dat
-s data/nrn_6.inp data/nrn_6.dat
-a .05 10 data/nrn_7.dat
-s data/nrn_7.inp data/nrn_7.dat
-a .05 10 data/nrn_8.dat
-s data/nrn_8.inp data/nrn_8.dat
-a .05 10 data/nrn_9.dat
-s data/nrn_9.inp data/nrn_9.dat
```

In this example, the command file propagates the model then takes a snapshot of the output and synaptic nodes. This is repeated until the simulation finishes.

B.4 Model Output and Post Processing

Figure B.2 shows how the output from our models branched off into various post processing image sequences. The adaptive model and O&G model both produced image sequences. Each set was processed using either a gaussian- windowed moving average or a level-windowed moving average. The result was a huge number of files. All these file were in ASC II form and had to be transfered into TIFF or EPS format to display the results. The software used to post process the

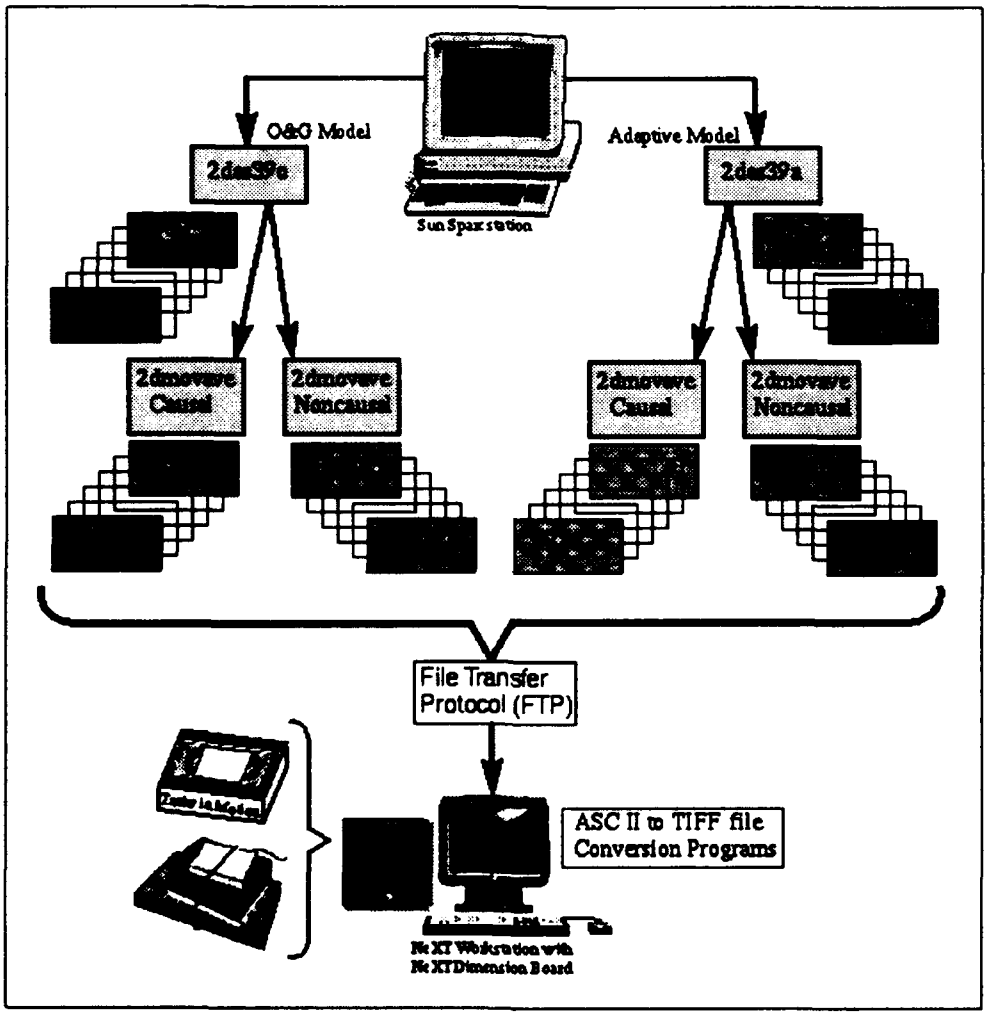

Figure B.2. This diagram shows the steps taken to prepared the model and images for simulation. The process involves several computer systems and source files.

files is included at the end of this section. We used the NeXT workstation to display our results. The Next provided facilities to convert the ASC II files as well as to import them into diagrams (such as the figures in this appendix).

The conversion program used to turn the ASC II data into TIFF files was called `make_tiff` and was a Cshell that called `float_gray` and `gra2tif` files out of the Utah_3.0 library. The problem with this process is that the `make_tiff` conversion looks at only a single image at a time. During the conversion, the minimum and maximum values in each image are found and all the pixel values in that image are scaled to these values so that the resultant image has maximum pixels at 255 and minimum values at zero. If you are using a sequence of images, each one may have different limiting pixels and when they are displayed as an animation using **Movieworks.app** (see Appendix D), the movie flickers. To fix this problem, we had to go through all the frames and find the global limiting values. Then artificially place these values in pixel location one and two (the location is arbitrary, they can be any where) in each frame. You can see these pixels in the enlarged figures of Chapter IV. The code used to do this operation is called `maxfind.c` and is included at the end of this section.

B.4.1 Post Processing Software: `2dmovave.c`

```
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main(int argc, char *argv[])
{
    FILE *outfileC, *outfileNC, *outfile1d1, *outfile1d2;
    FILE *infile1, *infile2, *infile3;
    FILE *infile4, *infile5, *infile6;
    FILE *infile7, *infile8, *infile9;
    char filename1[25], filename2[25], filename3[25];
    char filename4[25], filename5[25], filename6[25];
    char filename7[25], filename8[25], filename9[25];
    char number[25];
    char filenameOutC[25], filenameOutNC[25];
    float clevelout, nclevelout, levelin[10];
    float cave, ncave;
    int i,j,k,l, maxfiles;

    if(argc < 3 ) {
            printf("You need to provide an file root and the max. no. of files\n");
            exit(1);
    }

    maxfiles = atoi(argv[2]);
    for(i=8;i<maxfiles-4;i++) {

            strcpy(filename1,argv[1]);
            strcpy(filename2,argv[1]);
            strcpy(filename3,argv[1]);
            strcpy(filename4,argv[1]);
            strcpy(filename5,argv[1]);
            strcpy(filename6,argv[1]);
            strcpy(filename7,argv[1]);
            strcpy(filename8,argv[1]);
            strcpy(filename9,argv[1]);
            strcpy(filenameOutC,argv[1]);
            strcpy(filenameOutNC,argv[1]);
```

```
if(i<14){
        strcat(filename1,"00");
}
else {
        if(i<104){
                strcat(filename1,"0");
        }
}
if(i<13){
        strcat(filename2,"00");
}
else {
        if(i<103){
                strcat(filename2,"0");
        }
}
if(i<12){
        strcat(filename3,"00");
}
else {
        if(i<102){
                strcat(filename3,"0");
        }
}
if(i<11){
        strcat(filename4,"00");
}
else {
        if(i<101){
                strcat(filename4,"0");
        }
}
if(i<10){
        strcat(filename5,"00");
        strcat(filenameOutC,"00");
        strcat(filenameOutNC,"00");
}
else {
        if(i<100){
                strcat(filename5,"0");
        }
}
if(i<9){
        strcat(filename6,"00");
}
else {
        if(i<99){
                strcat(filename6,"0");
        }
}
if(i<8){
        strcat(filename7,"00");
}
else {
        if(i<98){
                strcat(filename7,"0");
        }
}
if(i<7){
        strcat(filename8,"00");
}
else {
        if(i<97){
                strcat(filename8,"0");
        }
}
if(i<6){
        strcat(filename9,"00");
}
else {
```

```
        if(i<96){
                strcat(filename9,"0");
        }
}
sprintf(filename1,"%s%d%s",filename1,(i-4),".dat");
sprintf(filename2,"%s%d%s",filename2,(i-3),".dat");
sprintf(filename3,"%s%d%s",filename3,(i-2),".dat");
sprintf(filename4,"%s%d%s",filename4,(i-1),".dat");
sprintf(filename5,"%s%d%s",filename5,(i  ),".dat");
sprintf(filename6,"%s%d%s",filename6,(i+1),".dat");
sprintf(filename7,"%s%d%s",filename7,(i+2),".dat");
sprintf(filename8,"%s%d%s",filename8,(i+3),".dat");
sprintf(filename9,"%s%d%s",filename9,(i+4),".dat");
sprintf(filenameOutC,"%s%d%s",filenameOutC,(i),".cas");
sprintf(filenameOutNC,"%s%d%s",filenameOutNC,(i),".ncs");
printf("Openning Files %d - %d.\n", i-4, i+4);
printf("%s\n",filename1);
printf("%s\n",filename5);
printf("%s\n",filename9);
if ((infile1 = fopen(filename1, "r")) != NULL &&
        (infile2 = fopen(filename2, "r")) != NULL &&
        (infile3 = fopen(filename3, "r")) != NULL &&
        (infile4 = fopen(filename4, "r")) != NULL &&
        (infile5 = fopen(filename5, "r")) != NULL &&
        (infile6 = fopen(filename6, "r")) != NULL &&
        (infile7 = fopen(filename7, "r")) != NULL &&
        (infile8 = fopen(filename8, "r")) != NULL &&
        (infile9 = fopen(filename9, "r")) != NULL &&
        (outfileC = fopen(filenameOutC, "w")) != NULL &&
        (outfileNC = fopen(filenameOutNC, "w")) != NULL) {
        if((fscanf(infile1, "%f", &levelin[1])) != EOF &&
           (fscanf(infile2, "%f", &levelin[2])) != EOF &&
           (fscanf(infile3, "%f", &levelin[3])) != EOF &&
           (fscanf(infile4, "%f", &levelin[4])) != EOF &&
           (fscanf(infile5, "%f", &levelin[5])) != EOF &&
           (fscanf(infile6, "%f", &levelin[6])) != EOF &&
           (fscanf(infile7, "%f", &levelin[7])) != EOF &&
           (fscanf(infile8, "%f", &levelin[8])) != EOF &&
           (fscanf(infile9, "%f", &levelin[9])) != EOF) {
                l = 1;
                j = 1;
                do {
                        cave = 0.0;
                        for(k=1;k<5;k++)
                                cave += levelin[k];
                        ncave = cave;
                        cave /= 5.0;
                        for(k=6;k<9;k++)
                                ncave += levelin[k];
                        ncave /= 9.0;
                        clevelout = levelin[5]-cave;
                        nclevelout = levelin[5]-ncave;
                        if(l == 57) {
                                fprintf(outfileC,"%f\n ", clevelout);
                                fprintf(outfileNC,"%f\n ", nclevelout);
                                printf(".");
                                l = 1;
//                              printf("\n %d", l);
                                j++;
                        }
                        else {
                                fprintf(outfileC,"%f ", clevelout);
                                fprintf(outfileNC,"%f ", nclevelout);
//                              printf("%d", l);
                                l++;
```

```
                                        }
                                        } while((fscanf(infile1, "%f", &levelin[1])) != EOF &&
                                                (fscanf(infile2, "%f", &levelin[2])) != EOF &&
                                                (fscanf(infile3, "%f", &levelin[3])) != EOF &&
                                                (fscanf(infile4, "%f", &levelin[4])) != EOF &&
                                                (fscanf(infile5, "%f", &levelin[5])) != EOF &&
                                                (fscanf(infile6, "%f", &levelin[6])) != EOF &&
                                                (fscanf(infile7, "%f", &levelin[7])) != EOF &&
                                                (fscanf(infile8, "%f", &levelin[8])) != EOF &&
                                                (fscanf(infile9, "%f", &levelin[9])) != EOF);
                        }
                        fclose(outfileC);
                        fclose(outfileNC);
                        fclose(infile1);
                        fclose(infile2);
                        fclose(infile3);
                        fclose(infile4);
                        fclose(infile5);
                        fclose(infile6);
                        fclose(infile7);
                        fclose(infile8);
                        fclose(infile9);
                        printf("\n");
                }
                else {
                        printf("Couldn't Open File on Iteration: %d\n",i);
                        exit(1);
                }

        }
        return(0);
};
```

B.4.2 Post Processing Software: maxfind.c

```
#include <stdlib.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

main(int argc, char *argv[])
{
        FILE *outfile;
        FILE *infile1;
        char filename1[25], filenameOut[25], file[25], file1[25];
        float levelin1, levelin2, levelin, maxvalue, minvalue;
        int i, l, maxfiles;
        if(argc < 4 ) {
           printf("You need to provide a file root, the max. no. of files,\n
                           the input file extension and output file extension.\n");
           exit(1);
        }
        maxfiles = atoi(argv[2]);
        maxvalue = -255.0;
        minvalue = 255.0;
        for(i=8;i<maxfiles-4;i++) {
           strcpy(file,argv[1]);
           if(i<10) {
              strcat(file,"0");
           }
           if(i<100) {
               strcat(file,"0");
           }
           sprintf(filename1,"%s%d%s",file,i,argv[3]);
           printf("Openning File %d.\n", i);
           if((infile1 = fopen(filename1, "r")) != NULL) {
              fscanf(infile1, "%f", &levelin);
                  do {
                     if(levelin < minvalue)
                        minvalue = levelin;
```

```
                                    if(levelin > maxvalue)
                            maxvalue = levelin;
                        } while((fscanf(infile1, "%f", &levelin)) != EOF);
                        fclose(infile1);
                        printf("\n");
                    }
                }
//              printf("min: %f  max: %f\n",minvalue, maxvalue);
                for(i=8;i<maxfiles-4;i++) {
                    strcpy(file,argv[1]);
                    strcpy(file1,argv[1]);
                    if(i<10) {
                       strcat(file,"0");
                       strcat(file1,"0");
                    }
                    if(i<100) {
                       strcat(file,"0");
                       strcat(file1,"0");
                    }
                    sprintf(filename1,"%s%d%s",file,i,argv[3]);
                    sprintf(filenameOut,"%s%d%s",file1,i,argv[4]);
                    printf("Openning File %d to insert hi/lo values.\n", i);
                    if ((infile1 = fopen(filename1, "r")) != NULL &&
                        (outfile = fopen(filenameOut, "w")) != NULL) {
                        if(fscanf(infile1, "%f %f %f ", &levelin1, &levelin2, &levelin) != NULL) {
                            l = 3;
                            fprintf(outfile,"%f %f ", maxvalue, minvalue);
                            do {
                               fprintf(outfile,"%f ", levelin);
                               if(l == 64) {
                                  fprintf(outfile,"\n");
                                  printf(".");
                                  l = 1;
                               }
                               else {
                                  l++;
                               }
                            } while((fscanf(infile1, "%f", &levelin)) != EOF);
                        }
                        fclose(outfile);
                        fclose(infile1);
                        printf("\n");
                    }
                }
                return(0);
};
```

B.5 Animation and Output to Videotape

The files generated using **make_tiff** can be gathered together in a subdirectory and animated using the **Movieworks.app** program. We talk about this program in the next appendix under the pretext of image capture. Here we talk about its ability to animate. When the files are all in a sungle directory, that directory can be renamed with a ".mw" extension. Then, using **Movieworks.app** the directory can be *converted* (see the menu under files) into a movie. Some things to note. The files must begin with a zeroth image and be sequentially number to through to the last frame (**Do not** skip any numbers or the program will return and say that all the files are not of the same format. This is a poorly worded error to an inefficiency of the program.). The files must have

the same name as the subdirectory root with an underbar then the frame number following it. For example, the tank files we developed were put into a **tanka** subdirectory. We renamed it to **tanka.mw** and named all the files in it **tanka_##.tiff**. The results are worth the trouble as we were able to make some nice movies.

To output the movies, we used **Screenscape.app**. The program takes a portion of the screen and continually outputs it to the NeXTDimension board output ports in an NTSC format. Therefore, the only machine we could use was Bilbo as it was the only one with the board installed. We plugged a VHS recoder into the NeXTDimension board and recorded when we wanted output. **Screenscape.app** allows the user to identify a portion of the sceeen (in fixed mode) that will be relayed to the output ports. This is not a scalable region, it is fixed. The program will also allow you to have the output follow the cursor around. We did not use this mode (see thehelp in **Screenscape.app**). It is impossible to give a tutorial under this format so the reader will have to play around with the software mention herein.

B.5.1 Conversion Cshell: make_tiff

```
#!/bin/csh
if ($#argv != 3 ) then
echo "  ";
echo "The proper format is:"
echo " "
echo "make_tiff infile xsize ysize";
echo "  ";
exit -1
endif
set temp = $1:r
~/bin/float_gray $1 $temp.gra
~/bin/gra2tif  $2 $3 $temp.gra $temp.tiff
unalias rm
rm $temp.gra
source ${home}/.alias

#foreach f ($argv[3-] )
#echo "working on file: $f"
#end
```

Appendix C. **Simulation Source Code.**

The code listed in this appendix is the object oriented source code used to simulate a massively parallel neural network based on the neurotransmitter and cell-activity models. File headers (.hpp) are listed first followed by their source file (.cpp). The last section contains the source file for the *main()* routine.

C.1 Runge-Kutta Classes

C.1.1 RK.HPP This header file describes the RungeKutta base class and its two children classes – TransmitterLevel and CellActivity.

```
//*****************************
//
//  Base Class: RUNGE-KUTTA
//
//
//  Date: 31 May 92
//  Author: David Swanson
//
//  Revised: 15 July 92
//
//*****************************
class RungeKutta {
    // This base class is designed as a superclass to hold the attributes
    // and public methods for a Runge-Kutta numberical integrator.
    // This class implements a fourth-order Runge-Kutta (R-K) integration
    // technique.  Like the Euler method of solving dy/dx, R-K uses
    // a Taylor series expansion to approximate the value of y at x+dx.
    // Euler used the first two terms of the expansion giving a second-order
    // solution.  Fourth-order R-K gives an error of order O(h^5) and is the
    // most popular method for numerical solutions to differential Eqns.
    //
    // The equations used for R-K are;
    //
    //       y(x+dx) = y(x) + 1/6*[D1 + 2*D2 + 2*D3 + D4]
    //
    //  where,
    //
    //      D1 = dx * f(x        , y(x)        )
    //      D2 = dx * f(x + .5*dx, y(x) + .5*D1)
    //      D3 = dx * f(x + .5*dx, y(x) + .5*D2)
    //      D4 = dx * f(x +    dx, y(x) +    D3)
    //
    //      f(x,y) = dy/dx
    //
    // The R-K method is 'self-starting' meaning you don't need to know
    // anything about the history of the function to obtain the next step.
    // although the error is low, it can accumulate and give poor results
    // over the long run.  I will address this problem later.

    // DATA MEMBERS
private:    // These members are visible to this class only.
    float startY;   // Starting value of dependent variable.
    float nextY1,nextY2;   // Value of dependent variable after stepping the
                           // the independent variable dx.
    float deltaY;          // variable used to determine acceptability of answer
```

```
    float x;        // Initial value of independent variable.
    float dx;       // Interval over which integration is performed.
    float tolerance;       // This is the error value that is acceptable
    float D1, D2, D3, D4;  // Intermediate calcualtions

    // MEMBER FUNCTIONS
protected:  // This section contains member that are only accessable
            // to the members of this class and derived classes.

    // The following method is declared virtual float.  That is it will
    // be redefined in derived classes (virtual) and will return an address
    // to an array of floating point numbers as the result.  It accepts
    // the address of an array of floating point numbers for x and y.
    //
    // The zero at the end of the next line indicates that the virtual
    // member function is a 'pure' member.  It is only a place holder
    // and must be defined in all subclasses.  They can also define it
    // as a pure member.  This is required because the function is used
    // by an other member function in this class but the virtual function
    // is defined definitely in a child class.
    virtual float derivative(float, float *) = 0;
    // This member function is used to reduce the amount of code in
    // the propagateState() function.
    float rkAlgorithm(float, float, float *);
public:     // The members in this section will be seen by all objects in
            // the same scope as this class.
    // Because instantitions of this class need not be initialized or
    // destroyed, there is not a constructor or destructor for it.
    // The following member function is used to solve the differential
    // equation defined in 'derivative' given the initial conditions.
    void propagateState(float *);
};

//****************************
//
//  Derived Class: CellActivity
//
//  Date: 15 July 92
//  Author: David Swanson
//
//****************************
class CellActivity : public RungeKutta {
    // This class inherits the members of the Runge-Kutta class
    // and adds the specific member function called 'derivative'.
    float derivative(float, float *);
};

//****************************
//
//  Derived Class: TransmitterLevel
//
//  Date: 15 July 92
//  Author: David Swanson
//
//****************************
class TransmitterLevel : public RungeKutta {
    // This class inherits the members of the Runge-Kutta class
    // and adds the specific member function called 'derivative'.
    float derivative(float, float *);
};
```

C.1.2 RK.CPP The RungeKutta class members are defined in this source file.

```
// This module contains the members for classes;
//
//   RungeKutta
//       TranmsitterLevel
//
#include <stdio.h>
#include "RK.hpp"  // Class Header information
//***************************
//
//  Base Class: Runge-Kutta
//
//  Date: 31 May 92
//  Author: David Swanson
//
//  Revised: 15 July 92
//
//***************************
void RungeKutta::propagateState(float *param) {
// This member function of the Runge-Kutta base class (also an abstract
// class) impliments a variable step form of the fourth order RK algorithm.
// The member recieves data concerning the current value of the state, the
// desired step size, tolerance for the results, and input plus derivative
// parameters.  These parameters are contained in the parameter array and
// are passed as a pointer.
    //      param[0] = z
    //      param[1] = dt
    //      param[2] = Tolerance
    //      param[3] = I
    //      param[4 and up] are derivative constants
// The flow of this member is:
    // 1. Estimate the value of the state given a step size
    //    equivelent to the total time interval requested -> "nextY2".
    // 2. Estimate the value of the state given a step size
    //    half that of the requested time interval -> "nextY1".
    // 3. If the difference between nextY1 and nextY2 is greater than
    //    the tolerance requested, repeat 1 and 2 replacing nextY2 with
    //    nextY1.  Use the half the half interval for estimating nextY1.
    //    As so on.
    // 4. If the difference between nextY1 and nextY2 is less than
    //    the tolerance requested, use the larger step size to
    //    determine the true estimated value of the function.
    startY = param[0];
    tolerance = param[2];
    x = 0.0;
    // The purpose of the next DO loop is to find an appropriate
    // time interval that satisfies the desired error or tolerance.
    // Once the interval is identified a second do loop calculates
    // the the full interval change in state.
    do{
            dx = param[1] - x;
            do {
                    nextY2 = startY + rkAlgorithm(startY, dx, param);
                    dx = 0.5*dx;
                    nextY1 = startY;
                    nextY1 += rkAlgorithm(nextY1, dx, param);
                    nextY1 += rkAlgorithm(nextY1, dx, param);
                    deltaY = nextY2 - nextY1;
                    if(deltaY < 0.0)
                            deltaY = -deltaY;
            } while(deltaY > tolerance);
            x += 2.0*dx;
            startY = nextY1;
```

```
    } while(x < param[1]);
    param[0] = nextY1;
    if(2*dx == param[1])
            dx *= 4;
};
float RungeKutta::rkAlgorithm(float y0, float h, float *param) {
// This member function is designed to simplify the code in the
// propagate state member.
            D1 = h * derivative(y0       , param);
            D2 = h * derivative(y0+ .5*D1, param);
            D3 = h * derivative(y0+ .5*D2, param);
            D4 = h * derivative(y0+    D3, param);
            return (1.0/6.0) * (D1 + 2*D2 + 2*D3 + D4);
};
//***************************
//
//  Base Class: CellActivity
//
//  Date: 16 July 92
//  Author: David Swanson
//
//***************************
float CellActivity::derivative(float y, float *param) {
    // This is the cell activity derivative equation
    //
    // dxi/dt = Axi + B
    //
    //      param[0] = x
    //      param[1] = dt
    //      param[2] = Tolerance
    //      param[3] = I
    //
    //  param[4] -> Phi
    //  param[5] -> Psi
    //
    return param[4]*y + param[5];
};
//***************************
//
//  Base Class: TransmitterLevel
//
//  Date: 31  May 92
//  Author: David Swanson
//
//  Revised: 15 July 92
//
//***************************
float TransmitterLevel::derivative(float y, float *param) {
    // This is the neurotransmitter level for the synapse
    //
    //  dzi/dt = AB - (A+I)zi
    //
    //      param[0] = z
    //      param[1] = dt
    //      param[2] = Tolerance
    //      param[3] = I
    //  param[4] -> A
    //  param[5] -> B
    //
    return param[4]*param[5] - (param[4] + param[3])*y;
};
```

C.2 Neuron Classes

C.2.1 Neuron.HPP This section contains the header file for the Neuron base class and its children - Synapse and Soma.

```
//
//      Base Class:  Neuron (abstract)
//
//      Derived Classes: soma, synapse
//
class Amacrine;
//****************************
//
//  Base Class: NEURON
//
//
//  Date: 31 May 92
//  Author: David Swanson
//
//  Revised: 17 July 92
//
//****************************
// This is the base class for the neuron classes.  It contains the
// data members and member functions common to both the soma and
// synapse.
class Neuron {
       // Both the soma and synapse require knowledge of the current
       // activity level, upper and lower bounds, input stimulii, and
       // a decay constant.  The difference between the two is how their
       // responses are modeled.  This class holds the common information
       // and methods (member functions).
       // The first thing needed is a pointer to the Amacrine object
       // that instantiated this object.  This is a protected data member
       // because it will be needed by the child of this class but no
       // other objects need access.  The synapse and soma objects both
       // will send a request to propagate their internal state via the
       // Runge-Kutta state estimation engine also termed the propagator.

protected:
       // The following members are used to store various parameters
       // describing the response of the neuron and the current stimulus
       // and activity or neurotransmitter level of the neuron.  The
       // data members are protected and the member functions are public.
       float parameters[8]; // This array is used to transfer data to the
                                                    // Runge-Kutta estimator engines.
       // Each neuron must have an identity so these data members are used
       // to store the row and col location of the neuron in the Amacrine
       // arrays.
       int r, c;  // Here x is the column number and y is the row.
public:        // Member Functions
       // The next function is used to set the current stimulus level
  // The object must be instantiated so here are the
  // constructor and destructor functions.
       // Neuron();
       // ~Neuron();
       void stimulate(float stimulus) {parameters[3] = stimulus;};
       void delayedStimulus(float delayedValue, float delayedInput)
                                                          {parameters[7] = delayedValue;
                                                           parameters[8] = delayedInput;};
       void setPosition(int row, int col) {r = row; c = col;};
       // Other objects may be interested in the internal values of a neuron.
```

```
        // Therefore the report function hands over the address of the parameter
        // array.
        float *report(void) {return parameters;};
        // The following member functions will be redefined in children
        // classes.  They are used to instruct the neuron to determine
        // the output or respond and to report the current activity
        // or level.
        virtual void advance(float) = 0;     // Defined as a pure virtual
        virtual void steadyState(void) = 0; // To be defined later.
};

//****************************
//
//  Derived Class: Synapse
//
//
//  Date: 31 May 92
//  Author: David Swanson
//
//  Revised: 15 July 92
//
//****************************
// This class contains the data members and member functions used by
// a synapse.  Unlike the Soma, this class uses an equation of
// exponentials to determine the neurotransmitters level after a
// step in time.  This class represents the long term memory of the
// system.
class Synapse : public Neuron {
public:
        // The following constructor accepts the Synapse's location
        // as int, int and it also recieves a pointer to the RK state
        // propagator object classed as a Transmitter Level child of the
        // RungeKutta base class.
        Synapse(void);
        // ~Synapse();  I don't think a destructor is needed.
        void advance(float);
        void steadyState(void);
};


//****************************
//
//  Derived Class: Soma
//
//
//  Date: 31 May 92
//  Author: David Swanson
//
//  Revised: 15 July 92
//
//****************************
// This class contains the data members and member functions used by
// the neuron's soma to determine the current activity level of the cell.
// The response of this cell to applied stimulii follows the Ogmen
// differential equations describing short term memory.  The soma
// depends on the long term memory of the synapses from neighboring
// cells.  The heireditary affiliation of this class to the neuron
// must be public to maintain the protected members of the parent
// class.
class Soma : public Neuron {
        // Even though this class has an identity, the following
        // data members identify the size of the receptive field.
        // These members are used to calculate the neuron's activity
        // based on a differential equation.
```

```
private:
        float A;        // Natural decay term
        float B;        // Upper bound
        float D;        // Lower bound
        float sumZI;// A variable used to hold the summed IZ values.
        float ZI;       // A variable to hold the IZ value cooresponding
                                // to the soma cell location.

        // This function is used to gather the inhibitory inputs from
        // neighboring synapses.
        void sumZIvalues(void);
public:
        // When an object is created from this class, the current activity
        // must be determined.  The following constructor sets the data
        // members that define the neuron's response characteristic and the
        // initial stimulus.  Then, using the steady-state equations
        // that characterize the cell's response, it sets the current activity.
        Soma(void);
        // ~Soma();  I don't think I need a destructor.
        void advance(float);
        void steadyState(void);
};
```

C.2.2 asyneur.CPP The Synapse and Soma members are defined in this file.

```
// This module contains the members for classes;
//
//      Neuron
//      Soma
//       Synapse
//
#include <stdio.h>
#include <math.h>
#include "defines.h"
#include "rk.h"
#include "neuron.h"  // Class Header information
#include "amacrine.h"
extern Soma soma[(ROWS-2*FrSIZE)][(COLS-2*FcSIZE)];
extern Synapse synapse[ROWS][COLS];
extern TransmitterLevel synapseRK;
extern CellActivity somaRK;
extern Amacrine thePlexus;
//*****************************
//
//  Derived Class: Synapse
//
//  Date: 15 July 92
//  Author: David Swanson
//
//*****************************
Synapse::Synapse(void) {
        // After this class does invokes the constructor for the parent class it
        // must assign constant values that characterize its nature.
        parameters[0] =   0.0;      // z     (or transmitter level initial)
        parameters[1] =   0.1;      // dt
        parameters[2] =   TOLSYNAPSE;// Tolerance
        parameters[3] =   0.0;      // I     (or stimulii)
        parameters[4] =   SYN_ALPHA;// alpha (or natural decay term)
        parameters[5] =   SYN_BETA; // beta  (or rest value)
};
void Synapse::steadyState(void) {
```

```
        float a,b;

        a = parameters[4] * parameters[5];
        b = parameters[4] + parameters[3];
#ifdef ECHO
        printf("Synapse: %d, %d => %f\n",r,c,a/b);
#endif

        parameters[0] = a / b;
};
void Synapse::advance(float interval) {
        // This member function tells the synapse to propagate its state
        // value over the time interval specified.

        parameters[1] = interval;

        synapseRK.propagateState(parameters);
#ifdef ECHO
        printf("Synapse: %d, %d => %f\n",r,c,parameters[0]);
#endif
};
//****************************
//
//  Derived Class: Soma
//
//  Date: 23 July 92
//  Author: David Swanson
//
//****************************
// This constructor must alter the cell location x and y to indicate an
// offset from the boundary of the receptive field.
Soma::Soma(void) {
        parameters[0] =   0.0;       // x     (or cell activity - initial)
        parameters[1] =   0.1;       // dt
        parameters[2] =   TOLSOMA;    // Tolerance
        parameters[3] =   0.0;       // I     (or stimulii)
        parameters[4] =   0.0;       // Psi     (Coefficient 1)
        parameters[5] =   0.0;       // Phi     (Coefficient 2)

        A = SOMA_DECAY;                          // Natural Decay
        B = SOMA_UPPER_BOUND;      // Upper bound
        D = SOMA_LOWER_BOUND;                                    // Lower Bound
};
void Soma::steadyState(void) {
        float a,b;

        sumZIvalues();

        a = B*ZI - D*sumZI;
        b = A + sumZI + ZI;
#ifdef ECHO
        printf("Soma: %d, %d => %f\n",r,c,a/b);
#endif

        parameters[0] = a / b;
};
void Soma::advance(float interval) {
        float deltaTime;
        double fractional, dt, integer;
        int i;

        // This member function tells the synapse to propagate its state
        // value over the time interval specified.
        sumZIvalues();

        parameters[4] = -A - sumZI - ZI;       // Phi
        parameters[5] = B*ZI - D*sumZI;  // Psi

        deltaTime = PI/(2.0*(A+B*ZI-D*sumZI));  // Nyquist Sampling Rate

        if (interval > deltaTime) {
                dt = (double)interval/(double)deltaTime;
```

```
                fractional = modf(dt, &integer);
                parameters[1] = deltaTime;
                for(i=0;i<integer;i++)
                        somaRK.propagateState(parameters);
                interval = (float)fractional * deltaTime;
        }
        parameters[1] = interval;
        somaRK.propagateState(parameters);
#ifdef ECHO
        printf("Soma: %d, %d => %f\n",r,c,parameters[0]);
#endif
};
void Soma::sumZIvalues(void) {
        int i, i0, j, j0;
        float *param;
        sumZI = 0.0;
        ZI = 0.0;
        // To sum the ZI values in two dimensions, a set of nested
        // for loops are used starting at the low end of the receptive
        // field and spanning the a and y space.  Since the pointers
        // to the appropriate synapses are located in the amacrine object,
        // a series of indirections are used to send a message to report
        // the current values of the transmitter level.  Also contained
        // in the parameter set is the current stimulus.
        for(i=0;i<2*FrSIZE+1;i++){
                for(j=0;j<2*FcSIZE+1;j++) {
                        param = synapse[i+r][j+c].report();
#ifdef ASYNC
        #ifdef Ogmen
                        sumZI += (thePlexus.kernalI[i][j]) * param[8]; // sumZI*Gi
                        ZI    += (thePlexus.kernalE[i][j]) * param[3]; // ZI*Ge
        #else
                        sumZI += (thePlexus.kernalI[i][j]) * param[7] * param[8]; // sumZI*Gi
                        ZI    += (thePlexus.kernalE[i][j]) * param[0] * param[3]; // ZI*Ge
        #endif
#else
        #ifdef Ogmen
                        sumZI += (thePlexus.kernalI[i][j]) * param[3]; // sumZI*Gi
                        ZI    += (thePlexus.kernalE[i][j]) * param[3]; // ZI*Ge
        #else
                        sumZI += (thePlexus.kernalI[i][j]) * param[0] * param[3]; // sumZI*Gi
                        ZI    += (thePlexus.kernalE[i][j]) * param[0] * param[3]; // ZI*Ge
        #endif
#endif
                }
        }
};
```

C.3 Amacrine Classes

C.3.1 Amacrine.HPP

```
//****************************
//
//  Base Class: AMACRINE
//
//
//  Date: 31 May 92
//  Author: David Swanson
//
//****************************
// The controlling object for this simulation is the Amacrine plexus.
// This is a single object containg the neurons and synapses.  This class
// maintains the link between the neurons and the synapses.
class Amacrine {
```

```
        // This class mediates the operation of the simulation.
        //
        // It sends messages to the various objects that set the
        // current stimulus and step size.  The messages also tell
        // the neuron objects to respond and report.  Therefore, there
        // are two arrays of pointers to type Neurons.  One for synapses
        // and the other for soma.  The soma array is private as it is
        // only used by the amacrine object.  The synapse array must
        // be public so that soma objects can get the current neurotransmitter
        // level from each synapse.  Therefore, the soma objects must
        // have a pointer to the amacrine object to reference the
        // synapse array (I.E. myPlexus->synapse[?][?] where myPlexus is
        // the pointer to an Amacrine object).
private:
        FLAG somaDir, synDir;
        float maxStimuli;
        struct historicalData { float sampleTime;
                                                    struct historicalData *nextSample;
                                                    float synapseValues[ROWS][COLS];
                                                    float synapseInputs[ROWS][COLS];
                                                }*historyMostRecent,
                                                 *historyOldest;
        void forceSteadyState(void);
        void advancePlexus(float stepSize);
public:
        float kernalE[2*FrSIZE+1][2*FcSIZE+1], kernalI[2*FrSIZE+1][2*FcSIZE+1];
        // This object must be able to open the input image files and distribute
        // intensity data to the soma and synapse objects.  Then it must
        // be able to open output files to store current intensity data.
public:
        // Upon instantiation of the model, neurons must be created
        // and placed in the respective arrays.  The following member
        // functions are used for that purpose.  The first two functions
        // is this classes constructor and destructor.  The constructor
        // uses the add... functions which allocates memory and gets pointer
        // information.  The destructor deallocates the memory of all objects
        // in the arrays.
        Amacrine();
        ~Amacrine();
        // The last function needed for simulation purposes is the
        // runTimeStep function which steps through the two arrays
        // sending messages to reponds to the input applied.
        void runTimeStep(float stepSize, char *filename, FLAG steadystate);
        void gatherOutput(char *filenameSynapse, char *filenameSoma);
        void synapseSnapshot(int synRow, int synCol, float time, char *filenameSynapse);
        void somaSnapshot(int somaRow, int somaCol, float time, char *filenameSoma);
        void synapticPlaneSnap(char *filenameSynapse);
        void somaticPlaneSnap(char *filenameSoma);
        void synapseLinearSnap(int row, float time, char *filenameSynapse);
        void somaLinearSnap(int row, float time, char *filenameSoma);
};
```

C.3.2 asynamac.CPP

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#include "defines.h"
#include "rk.h"
#include "neuron.h"
#include "amacrine.h"

extern Soma soma[(ROWS-2*FrSIZE)][(COLS-2*FcSIZE)];
extern Synapse synapse[ROWS][COLS];
```

```
extern TransmitterLevel synapseRK;
extern CellActivity somaRK;
//****************************
//
//  Base Class: Amacrine
//
//  Date: 23 July 92
//  Author: David Swanson
//
//****************************
Amacrine::Amacrine() {
        // This constructor must create the synapsis and soma arrays and
        // fill them.
        int i, j, di, dj;
        FILE *outfile;
        double exponent, scalar, sigmaCol;
#if (FrSIZE != 0)
        double sigmaRow;
#endif
        somaDir = YES;
        synDir = YES;
        maxStimuli = 0.0;
        historyMostRecent = NULL;
        historyOldest = NULL;
        sigmaCol = ((double)2.0*(double)(FcSIZE+1.0))/(double)7.0;
#if (FrSIZE != 0)
        sigmaRow = ((double)2.0*(double)(FrSIZE+1.0))/(double)7.0;
        scalar = (double)1.0/(sigmaRow*sigmaCol*(double)(2.0*PI));
#else
        scalar = (double)1.0/(sigmaCol*sqrt((double)(2.0*PI)));
#endif
        for(i=0;i<2*FrSIZE+1;i++) {
                for(j=0;j<2*FcSIZE+1;j++) {
                        exponent = pow(((double)(j-FcSIZE)/sigmaCol),(double)2.0);
#if (FrSIZE != 0)
                        exponent += pow(((double)(i-FrSIZE)/sigmaRow),(double)2.0);
#endif
                        exponent *= -(double)0.5;
                        if((float)(scalar*exp(exponent)) < 0.000001)
                                kernalI[i][j] = 0.0;
                        else
                                kernalI[i][j] = (float)(scalar*exp(exponent));
                }
        }
        sigmaCol = ((double)2.0*(double)(EcSIZE+1.0))/(double)7.0;
#if (FrSIZE != 0)
        sigmaRow = ((double)2.0*(double)(ErSIZE+1.0))/(double)7.0;
        scalar = (double)1.0/(sigmaRow*sigmaCol*(double)(2.0*PI));
#else
        scalar = (double)1.0/(sigmaCol*sqrt((double)(2.0*PI)));
#endif
        for(i=0;i<2*FrSIZE+1;i++) {
                for(j=0;j<2*FcSIZE+1;j++) {
                        exponent = pow(((double)(j-FcSIZE)/sigmaCol),(double)2.0);
#if (FrSIZE != 0)
                        exponent += pow(((double)(i-FrSIZE)/sigmaRow),(double)2.0);
#endif
                        exponent *= -(double)0.5;
                        if((float)(scalar*exp(exponent)) < 0.000001)
                                kernalE[i][j] = 0.0;
                        else
                                kernalE[i][j] = (float)(scalar*exp(exponent));
                }
        }
        for(i=0;i<ROWS;i++)
                for(j=0;j<COLS;j++)
                        synapse[i][j].setPosition(i,j);
```

```
        for(i=0;i<ROWS-2*FrSIZE;i++)
                for(j=0;j<COLS-2*FcSIZE;j++)
                        soma[i][j].setPosition(i,j);
};
Amacrine::~Amacrine(){
        // This destructor must release the memory held by the cells
        int i, j;
        struct historicalData *tmpPtr;
#ifdef ASYNC
        while(historyOldest != historyMostRecent) {
                tmpPtr = historyOldest->nextSample;
                delete historyOldest;
                historyOldest = tmpPtr;
        }
        delete historyMostRecent;
#endif
};

void Amacrine::runTimeStep(float stepSize, char *filename, FLAG steadystate) {
        // This member function is designed to apply the stimulus
        // to each synapse object as given in the file "filename".
        // This file must be a file with gray-values arranged one value
        // per line (followed by a carriage return), in row by column
        // form.  The number of rows and columns must match the defined
        // ROWS and COLS values given above.
        //
        // After each synapse gets its stimulus, the soma
        // objects are instructed to propagate over the time step
        // supplied.  Once all the soma objects are complete, the
        // synapse objects are told to propagate.
        char grayLevel[25];
        int i, j;
        FILE *infile;
        float deltaTime, stimuli;
        double dt, fractional, integer;

        if ((infile = fopen(filename, "r")) != NULL) {
                maxStimuli = 0.0;
                for(i=0;i<ROWS;i++) {
                        for(j=0;j<COLS;j++) {
                                fscanf(infile, "%s", grayLevel);
#ifdef ECHO
                                printf("GrayLevel Input[%d][%d]: %s RAM: near - %lu far - %lu}\n",
i,j,grayLevel,coreleft(),farcoreleft());
#endif
                                stimuli = atof(grayLevel)/63.75;  // 63.75 is a scaling factor used in
                                                                  // in the tank simulations
                                if (stimuli > maxStimuli)
                                        maxStimuli = stimuli;
                                synapse[i][j].stimulate(stimuli);
                        }
                }
                fclose(infile);
        }
        if(steadystate == YES)
                forceSteadyState();
        else {
                deltaTime = PI/(2.0*(SYN_ALPHA+maxStimuli));  // Nyquist Sampling Rate
                if (stepSize > deltaTime) {
                        dt = (double)stepSize/(double)deltaTime;
                        fractional = modf(dt, &integer);
                        for(i=0;i<integer;i++)
                                advancePlexus(deltaTime);
                        stepSize = (float)fractional * deltaTime;
                }
```

```
                advancePlexus(stepSize);
        }
};
void Amacrine::forceSteadyState(void) {
        int i, j;
        float *paramPtr;
        struct historicalData *tmpPtr;
#ifdef ASYNC
        while (historyOldest != NULL) {
                tmpPtr = historyOldest->nextSample;
#ifdef ECHO
                printf("Deleting A History Cell in ForceSteadyState\n");
#endif
                delete historyOldest;
                historyOldest = tmpPtr;
        }
        if(!(historyMostRecent = new struct historicalData)) {
                printf("Not enough RAM for an additional historicalData structure");
                exit(1);
        }
        historyOldest = historyMostRecent;
#endif
        for(i=0;i<ROWS;i++) {
                printf("[%2d] ",i);
                for (j=0;j<COLS;j++) {
                        synapse[i][j].steadyState();
                        printf(".");
#ifdef ECHO
                                printf("steadyState: synapse[%d][%d]\n", i,j);
#endif
#ifdef ASYNC
                        paramPtr = synapse[i][j].report();
                        historyMostRecent->synapseValues[i][j] = paramPtr[0];
                        historyMostRecent->synapseInputs[i][j] = paramPtr[3];
                        synapse[i][j].delayedStimulus(paramPtr[0],paramPtr[3]);
#endif
                }
                printf("\t Synapse\n");
        }
#ifdef ASYNC
        historyMostRecent->nextSample = NULL;
        historyMostRecent->sampleTime = 0.0;
#endif
        for(i=0;i<ROWS-2*FrSIZE;i++) {
                printf("[%2d] ",i);
                for (j=0;j<COLS-2*FcSIZE;j++) {
                        soma[i][j].steadyState();
                        printf(".");
#ifdef ECHO
                                printf("steadyState: soma[%d][%d]\n",i,j);
#endif
                        }
                printf("\t Soma\n");
        }
};
void Amacrine::advancePlexus(float stepSize) {
        int i, j;
        float targetTime;
        struct historicalData *tmpPtr, *lastPtr;
        float *paramPtr;
#ifdef ASYNC
        if(historyMostRecent == NULL) {
                printf("You must run force steady state using the -i option first");
                exit(1);
        }
        targetTime = (historyMostRecent->sampleTime) - DELAY_TIME;
        tmpPtr = historyOldest;
        lastPtr = tmpPtr;
```

```
        while((tmpPtr != NULL) && (tmpPtr->sampleTime <= targetTime)) {
                lastPtr = tmpPtr;
                tmpPtr = tmpPtr->nextSample;
        }
        for(i=0;i<ROWS;i++)
                for (j=0;j<COLS;j++)
                        synapse[i][j].delayedStimulus(lastPtr->synapseValues[i][j],
                                                                        lastPtr->synapseInputs[i][j]);
        while(historyOldest != lastPtr) {
                tmpPtr = historyOldest->nextSample;
        #ifdef ECHO
                printf("Deleting A History Cell in Advance\n");
        #endif
                delete historyOldest;
                historyOldest = tmpPtr;
        }
#endif
        for(i=0;i<ROWS-2*FrSIZE;i++)
                for (j=0;j<COLS-2*FcSIZE;j++)
                        soma[i][j].advance(stepSize);
#ifdef ASYNC
        tmpPtr = historyMostRecent;
        if(!(historyMostRecent = new struct historicalData)) {
                printf("Not enough RAM for an additional historicalData structure");
                exit(1);
        }
        tmpPtr->nextSample = historyMostRecent;
#endif
        for(i=0;i<ROWS;i++)
                for (j=0;j<COLS;j++) {
                        synapse[i][j].advance(stepSize);
#ifdef ASYNC
                        paramPtr = synapse[i][j].report();
                        historyMostRecent->synapseValues[i][j] = paramPtr[0];
                        historyMostRecent->synapseInputs[i][j] = paramPtr[3];
#endif
                }
#ifdef ASYNC
        historyMostRecent->sampleTime = (tmpPtr->sampleTime) + stepSize;
#endif
};

void Amacrine::gatherOutput(char *filenameSynapse, char *filenameSoma) {
        // This member function poles all synapse and soma objects for their
        // current state and stores these values in the appropriate file
        // in a row by column format of floats separated by spaces.  This
        // Function is intended to provide data for image files.
        FILE *outfile;
        float *paramPtr;
        int i, j;
        if ((outfile = fopen(filenameSynapse, "w")) != NULL) {
                for(i=0;i<ROWS;i++) {
                        for(j=0;j<COLS;j++) {
                                paramPtr = synapse[i][j].report();
                                fprintf(outfile,"%10.6f ",paramPtr[0]);
                        }
                        fprintf(outfile,"\n");
                }
                fclose(outfile);
        }
        if ((outfile = fopen(filenameSoma, "w")) != NULL) {
                for(i=0;i<ROWS-2*FrSIZE;i++) {
                        for(j=0;j<COLS-2*FcSIZE;j++) {
                                paramPtr = soma[i][j].report();
                                fprintf(outfile,"%10.6f ",paramPtr[0]);
                        }
                        fprintf(outfile,"\n");
                }
```

```
                fclose(outfile);
        }
};
void Amacrine::synapseSnapshot(int synRow, int synCol, float time, char
*filenameSynapse) {
        // This member function poles a single synapse object for its
        // current state and stores the values in the appropriate file
        // with row location, column location, and object state on a line.
        // This function is intended to provided data to GNU 3-d plots.
        FILE *outfile;
        float *paramPtr;
        if ((outfile = fopen(filenameSynapse, "a")) != NULL) {
                paramPtr = synapse[synRow][synCol].report();
                fprintf(outfile,"%6.3f %10.6f\n", time, paramPtr[0]);
                fclose(outfile);
        }
        else {
                exit(1);
        }
};

void Amacrine::somaSnapshot(int somaRow, int somaCol, float time, char
*filenameSoma) {
        // This member function poles all soma objects for their
        // current state and stores these values in the appropriate file
        // with row location, column location, and object state on a line.
        // This function is intended to provided data to GNU 3-d plots.
        FILE *outfile;
        float *paramPtr;
        if ((outfile = fopen(filenameSoma, "a")) != NULL) {
                paramPtr = soma[somaRow][somaCol].report();
                fprintf(outfile,"%6.3f %10.6f\n", time, paramPtr[0]);
                fclose(outfile);
        }
        else {
                exit(1);
        }
};

void Amacrine::synapticPlaneSnap(char *filenameSynapse) {
        // This member function poles all synapse objects for their
        // current state and stores these values in the appropriate file
        // with row location, column location, and object state on a line.
        // This function is intended to provided data to GNU 3-d plots.
        FILE *outfile;
        float *paramPtr;
        int i, j;
        if ((outfile = fopen(filenameSynapse, "w")) != NULL) {
                for(i=0;i<ROWS;i++) {
                        for(j=0;j<COLS;j++) {
                                paramPtr = synapse[i][j].report();
                                fprintf(outfile,"%10.6f", paramPtr[0]);
                        }
#ifdef GNUPLOT
                        i++;
                        for(j=COLS-1;j>=0;j--) {
                                paramPtr = synapse[i][j].report();
                                fprintf(outfile,"%10.6f", paramPtr[0]);
                        }
#endif
                        fprintf(outfile,"\n"),
                }
                fclose(outfile);
        }
        else {
                exit(1);
```

```
        }
};

void Amacrine::somaticPlaneSnap(char *filenameSoma) {
        // This member function poles all soma objects for their
        // current state and stores these values in the appropriate file
        // with row location, column location, and object state on a line.
        // This function is intended to provided data to GNU 3-d plots.
        FILE *outfile;
        float *paramPtr;
        int i, j;
        if ((outfile = fopen(filenameSoma, "w")) != NULL) {
                for(i=0;i<ROWS-2*FrSIZE;i++) {
                        for(j=0;j<COLS-2*FcSIZE;j++) {
                                paramPtr = soma[i][j].report();
                                fprintf(outfile,"%10.6f", paramPtr[0]);
                        }
#ifdef GNUPLOT
                        i++;
                        for(j=COLS-2*FcSIZE-1;j>=0;j--) {
                                paramPtr = soma[i][j].report();
                                fprintf(outfile,"%10.6f", paramPtr[0]);
                        }
#endif
                        fprintf(outfile,"\n");
                }
                fclose(outfile);
         }
         else {
                exit(1);
         }
};
void Amacrine::synapseLinearSnap(int row, float time, char *filenameSynapse) {
        // This member function poles all synapse and soma objects for their
        // current state and stores these values in the appropriate file
        // with row location, column location, and object state on a line.
        // This function is intended to provided data to GNU 3-d plots.
        FILE *outfile;
        float *paramPtr;
        int i, j;
        if ((outfile = fopen(filenameSynapse, "a")) != NULL) {
#ifdef GNUPLOT
                if (synDir == YES) {
#endif
                        for(j=0;j<COLS;j++) {
                                paramPtr = synapse[row][j].report();
                                fprintf(outfile,"%d %6.3f %10.6f %10.6f %10.6f %10.6f\n", j, time,
paramPtr[0], paramPtr[3], paramPtr[7], paramPtr[8]);
                        }
#ifdef GNUPLOT
                        synDir = NO;
                }
                else {
                        for(j=COLS-1;j>=0;j--) {
                                paramPtr = synapse[row][j].report();
                                fprintf(outfile,"%d %6.3f %10.6f %10.6f %10.6f %10.6f\n", j, time,
paramPtr[0], paramPtr[3], paramPtr[7], paramPtr[8]);
                        }
                        synDir = YES;
                }
#endif
                        fclose(outfile);
        }
        else {
                exit(1);
        }
};
```

```
void Amacrine::somaLinearSnap(int row, float time, char *filenameSoma) {
        // This member function poles all synapse and soma objects for their
        // current state and stores these values in the appropriate file
        // with row location, column location, and object state on a line.
        // This function is intended to provided data to GNU 3-d plots.
        FILE *outfile;
        float *paramPtr;
        int i, j;
        if ((outfile = fopen(filenameSoma, "a")) != NULL) {
#ifdef GNUPLOT
                if (somaDir == YES) {
                        for(j=0;j<COLS-2*FcSIZE;j++) {
                                paramPtr = soma[row][j].report();
                                fprintf(outfile,"%d %6.3f %10.6f\n", j+FcSIZE, time, paramPtr[0]);
                        }
                        somaDir = NO;
                }
                else {
                        for(j=COLS-2*FcSIZE-1;j>=0;j--) {
                                paramPtr = soma[row][j].report();
                                fprintf(outfile,"%d %6.3f %10.6f\n", j+FcSIZE, time, paramPtr[0]);
                        }
                        somaDir = YES;
                }
#else
                for(j=0;j<COLS-2*FcSIZE;j++) {
                        paramPtr = soma[row][j].report();
                        fprintf(outfile,"%d %6.3f %10.6f\n", j+FcSIZE, time, paramPtr[0]);
                }
#endif
                fclose(outfile);
        }
        else {
                exit(1);
        }
};
```

C.4 Main Source – Amasim.CPP

This section holds the source code for the *main()* routine.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <string.h>
#include <dos.h>        // Dos only needed for 'sound' and 'delay'
#include <alloc.h>      // Dos only needed for 'coreleft'
#include "defines.hpp"
#include "RK.hpp"
#include "Neuron.hpp"
#include "Amacrine.hpp"
Soma huge soma[ROWS-2*FrSIZE][COLS-2*FcSIZE];
Synapse huge synapse[ROWS][COLS];
TransmitterLevel far synapseRK;
CellActivity far somaRK;
Amacrine far thePlexus;
main(int argc, char *argv[])
{
    float timeStep, time;
    int i, j, iterations, entries;
    FILE *outfile, *infile;
    char filename[25], filename0[25],
                 filename1[25], filename2[25], stimulusFile[25];
    char test[25], option[25];
    FLAG finished;
```

```
// This file creates an instance of the Amacrine Plexus.
//
// It supplies the plexus with a file with input values
// and instructs the plexus to propagate over a timestep.
//
// Then it gathers the current state values of the cells
// in two files.
if(argc < 2 ) {
        printf("You need to provide a simulation run file\n");
        exit(1);
}
if ((infile = fopen(argv[1], "r")) != NULL) {
    fscanf(infile, "%s", filename1);   // Synapse output and Input file
    fscanf(infile, "%s", filename2);   // Soma output file
    // Make sure the files are empty!
    if ((outfile = fopen(filename1, "w")) != NULL)
       fclose(outfile);
    else
        exit(1);
    if ((outfile = fopen(filename2, "w")) != NULL)
        fclose(outfile);
    else
        exit(1);
    finished = NO;
    do {
       if((fscanf(infile, "%s", option)) != EOF) {
          printf("{RAM available: %lu Bytes}\t", coreleft());
          switch (option[1]) {
             case 'i' :   // Initialize
             case 'I' :   // "I" or "i"
                  fscanf(infile, "%f %d %s",&timeStep,&iterations,stimulusFile);
                  printf("%s %f %d %s\n",option,timeStep,iterations,stimulusFile);
                  thePlexus.runTimeStep(0.0, stimulusFile, YES);
                  time = 0.0;
//                thePlexus.somaLinearSnap(0, time, filename2);
//                thePlexus.synapseLinearSnap(0, time, filename1);
                  break;
            case 'a' :  // Advance
            case 'A' :    // "a" or "A"
                  fscanf(infile, "%f %d %s",&timeStep,&iterations,stimulusFile);
                  printf("%s %f %d %s\n",option,timeStep,iterations,stimulusFile);
                  printf("0 ");
                  time += timeStep;
                  thePlexus.runTimeStep(timeStep, stimulusFile, NO);
//                thePlexus.somaLinearSnap(0, time, filename2);
//                thePlexus.synapseLinearSnap(0, time, filename1);
                  for(j=1;j<iterations;j++) {
                      printf("%d ", j);
                      time += timeStep;
                      thePlexus.runTimeStep(timeStep, NULL, NO);
//                    thePlexus.somaLinearSnap(0, time, filename2);
//                    thePlexus.synapseLinearSnap(0, time, filename1);
                  }
                  printf("\n");
                  break;
            case 's' : // Snapshot
            case 'S' : // "S" or "s"
                  fscanf(infile, "%s %s", filename0, filename);
                  printf("%s %s %s\n", option, filename0, filename);
                  thePlexus.somaticPlaneSnap(filename);
                  thePlexus.synapticPlaneSnap(filename0);
                  break;
            case 'p' : // Plane Snapshot
            case 'P' : // "P" or "p"
                  fscanf(infile, "%s %s", filename, filename0);
                  printf("%s %s %s\n", option, filename, filename0);
```

```
                    thePlexus.somaticPlaneSnap(filename);
                    thePlexus.synapticPlaneSnap(filename0);
                    break;
                default :
                    finished = YES;
            }
        }
        else {
            finished = YES;
        }
    } while (finished == NO);
    fclose(infile);
  }
  return(0);
};
```

Appendix D. **Working with Images**

D.1 Image Capture

Without a doubt, motion detection involves the analysis of sequences of still images. To present these images to a model simulation will require capturing the images *a priori* and applying them to the input nodes of the model as required. This section documents our efforts to create a set of image sequences from video inputs. First we looked at using video as a source of data.

D.1.1 MovieWorks. We found two programs that support video capture on the NeXT computer systems. The first was an application called MovieWorks. This program uses the Digital Eye conversion device connected to the NeXT through its DSP port to digitize incoming video signals.

Before launching the MovieWorks application, you need to connect a video camera or video cassette recorder (VCR) to the Digital Eye digitizer box. This box is connected to the NeXT workstation labeled **frodo**. MovieWorks is located in the **LocalApps** folder. Figure D.1 shows what the MovieWorks interface windows look like.The main panel, "Movie Recorder", has the same

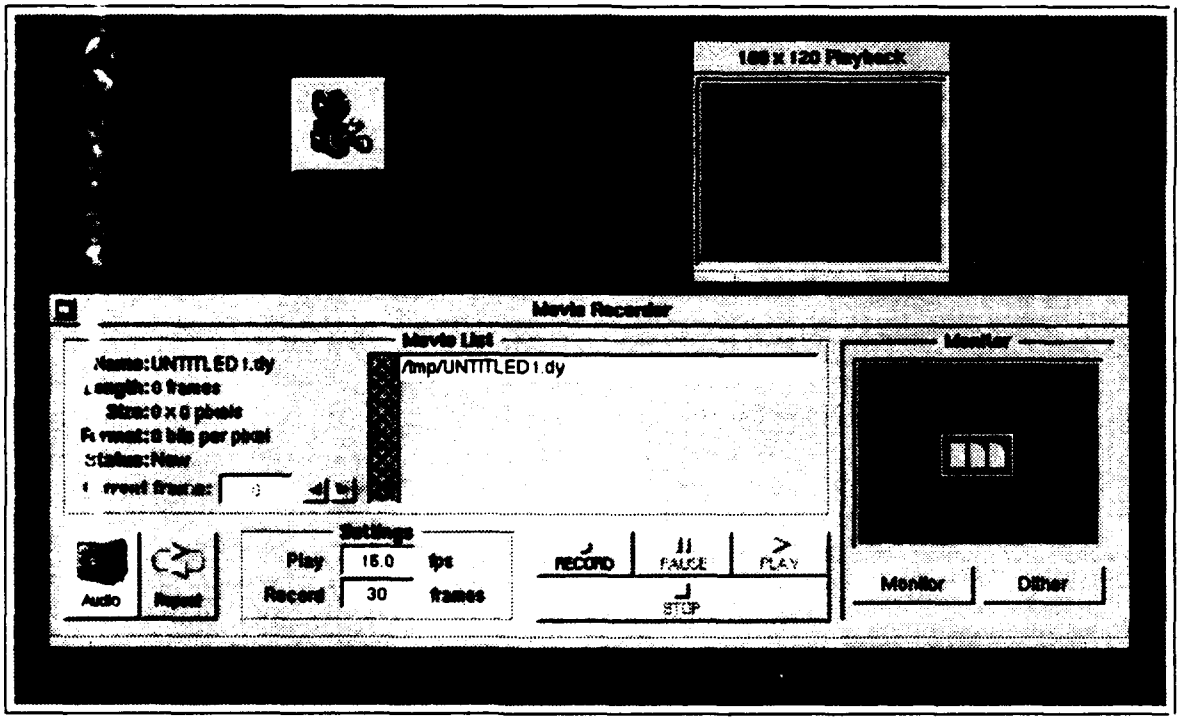

Figure D.1. MovieWorks Application Panels

buttons as would be seen on a VCR or camera. The monitor portion of this panel shows the current input as seen by the NeXT DSP port after the *monitor* button has been pressed. If the camera or

VCR is connected incorrectly or the video isn't running, the message "NO SIGNAL" appears in the monitor window. The Digital Eye digitizer box has two knobs that allow you to adjust the *black* and *white* levels of the picture. Any changes made using these knobs will be seen in the monitor window immediately. This application will allow you to capture as many as 15 frames per second (fps) and is user definable. Also defined by the user is the number of total frames to be captured in a session. Once these values are entered and you've started your video, click on the **Record** button to begin capture. After the frames are gathered MovieWorks prompts you for a filename before returning tot he Movie Recorder panel.

The documentation for MovieWorks states that the frames are stored using 4-bits per pixel and can be changed to 2-bits per pixel. Once the *movie* has been captured, you can review it by opening a movie file using the menu provided in the upper left hand corner of the screen and pressing the **Play** button. MovieWorks will display the movie in the "Playback" panel. If the **Repeat** button is selected, the movie will run continuously until the **Stop** button is pressed. Unfortunately, the resolution at 4-bits per pixel is substantially less than we expect to process. Therefore we decided to look for a different method of video capture.

D.1.2 VideoApps. The next method we tried was the VideoApps program. This program offered substantially better resolution than did MovieWorks due to its use of the NeXT Dimension (NXDimension) board installed in the **bilbo** NeXT workstation. Bilbo has a 21-inch color Megapixel monitor in addition to the regular monochrome monitor. The NXDimension board is installed internally in the NeXT cube and has its ports located in the back of the cube. There are seven ports to choose from. Four are labeled output and three are input. We used a Super VHS (SVHS) camera for video input and so we used the SVHS data jack. The other two ports are for regular RCA data jacks. These are the lower three ports. We had no use for the output ports but believe that one could use them to send data images to the VCR or camera for storage. There are two RCA, one SVHS, and one multipin D-connector available for output.

Once the camera was connected, We opened the VideoApps program located in the **Demos** folder. Another name for this application is NeXTV (Figure D.2). The difference between the two is the interface they use. We preferred to work with the VideoApps version (Figure D.3). VideoApps allows you to view the incoming video as did MovieWorks. The images displayed in the *CustomVideoView* portion of the panel. In the original version of VideoApps you could only *grab* a single frame at a time by pressing the **Grab** button. This resulted in consecutive frames separated by more than a minute due to the process time required to grab, compress, and save the data. Fortunately the source code is available for this application.

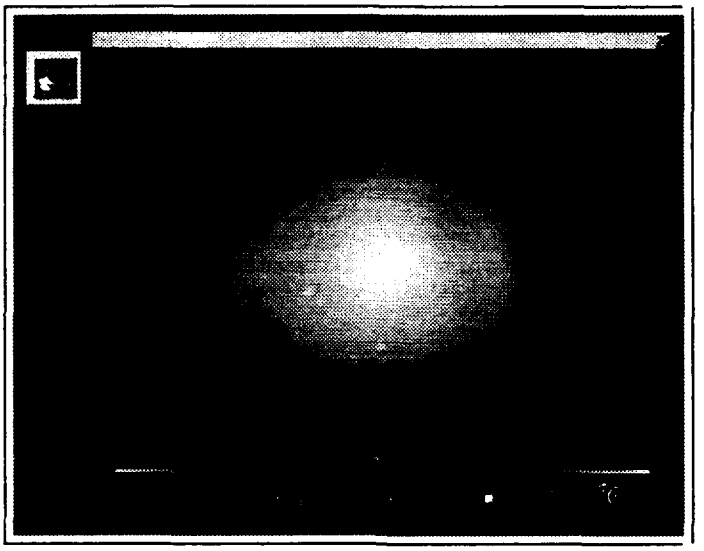

Figure D.2. NeXTV Application Panel

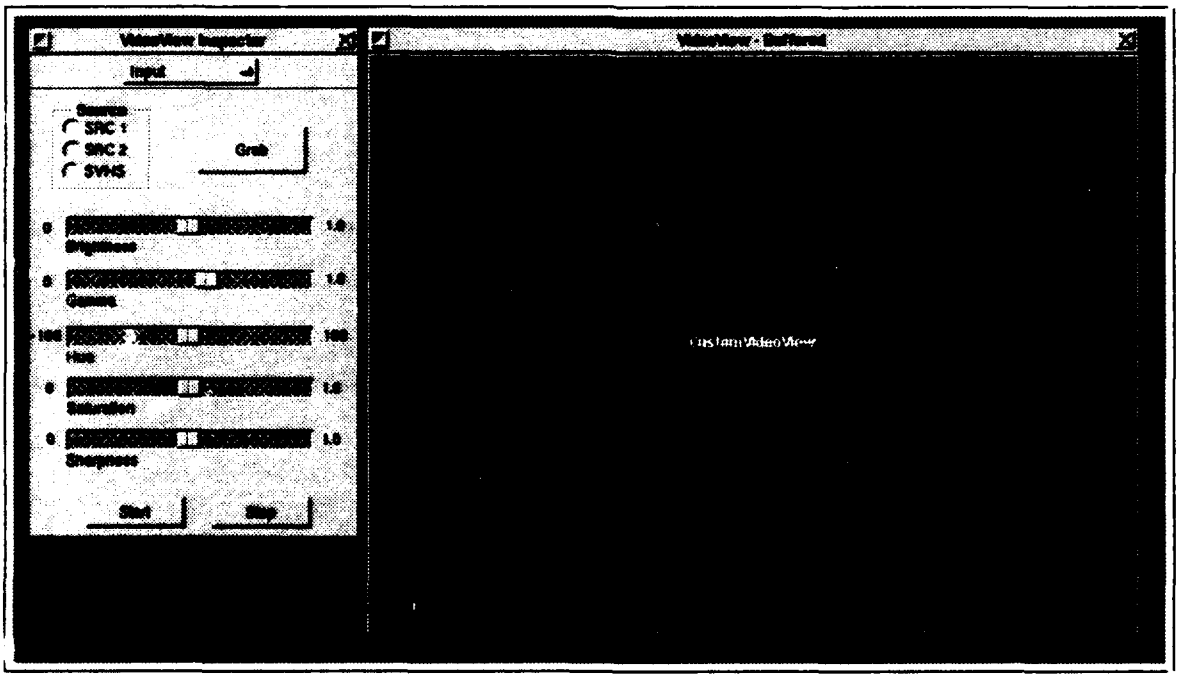

Figure D.3. VideoApps Application Panels

Coded in Objective-C, VideoApps is a simple collection of video objects written specifically for the NXDimension board. With the help of Captain Dennis Ruck, We located the grab object. The original version of this object looked like this:

```
- grab:sender
{
  id bitmap;
  id image;
  // Read the bits from the window
  image = [self grab];
  [image lockFocus];
  bitmap = [[NXBitmapImageRep alloc] initData:NULL
             fromRect:(grabRect.size.width == 0) ? &bounds
:&grabRect];
  [image unlockFocus];
  if (bitmap)
  {
    id mySavePanel = [SavePanel new];
    char filename[MAXPATHLEN+1];
    [mySavePanel runModal];
    if([mySavePanel filename])
    {
      NXStream *s = NXOpenMemory (NULL, 0, NX_READWRITE);
      strcpy(filename,[mySavePanel filename]);
      if(strcmp(&filename[strlen(filename)-5],".tiff"))
        strcat(filename,".tiff");
      if (s)
      {
        [bitmap
writeTIFF:susingCompression:NX_TIFF_COMPRESSION_JPEG
                 andFactor:10];
        NXFlush (s);
        if (NXSaveToFile (s, filename))
          NXRunAlertPanel("Error Saving File","Filename:
%s","OK",
                          NULL,NULL,filename);
        NXCloseMemory (s, NX_FREEBUFFER);
      }
    }
    [bitmap free];
  }
  return self;
}
```

By placing two simple *for(i=0;i<frames;i++)* loops in the code, We were able to improve the performance of the code. The resulting code looked like:

```
- grab:sender
{
  id bitmap[30], image;  // Declaring 30 bitmap objects
  int i, frames = 30;    // Setting the nomber of frames to grab
  char string[10];
  // Read the bits from the window
 // The {\it for} loop below is used to grab a number of
 // images equal to 'frames'.  Each image is an object into
 // which a snapshot of the video is placed using the lockfocus.
 // The image is transfered to a bitmap object which will be
 // saved later to disk.
```

```
  for(i=0;i<frames;i++) {
    image = [self grab];
    [image lockFocus];
    bitmap[i] = [[NXBitmapImageRep alloc] initData:NULL
        fromRect:(grabRect.size.width == 0) ? &bounds :
&grabRect];
    [image unlockFocus];
  }
 //  This segment of code asks the user to give a root filename
 // for the 30 image to follow.
  if (bitmap[i]){
     id mySavePanel = [SavePanel new];
     char filename[MAXPATHLEN+1];
     [mySavePanel runModal];
 // Using the root filename this segment of code saves the
 // bitmaps in separate files using the root filename plus a
 // numeric suffix to preserve the sequence.
     for(i=0;i<frames;i++) {
       if([mySavePanel filename]) {
         NXStream *s = NXOpenMemory (NULL, 0, NX_READWRITE);
         strcpy(filename,[mySavePanel filename]);
         if(strcmp(&filename[strlen(filename)-5],".tiff")) {
            sprintf(string,"%d",i);
            strcat(filename,string);
            strcat(filename,".tiff");
         }
         if (s) {
            [bitmap[i] writeTIFF:s
usingCompression:NX_TIFF_COMPRESSION_JPEG
                      andFactor:10];
            NXFlush (s);
            if (NXSaveToFile (s, filename))
                NXRunAlertPanel("Error Saving File","Filename:
                  %s","OK",NULL,NULL,filename);
            NXCloseMemory (s, NX_FREEBUFFER);
          }
       }
       [bitmap[i] free];
     }
  }
  return self;
}
```

Using a watch as the video subject (Figure D.4), we recorded 30 frames and calculated the capture rate. This turned out to be 1.8 seconds per frame.Although this is a 30 fold improvement over the original program, we believe that it will still be far too slow for my applications. Further improvements might be obtained if a direct input to stream capture method where incorporated, however, we decided to look for a different method of image capture. We learned of a system owned and operated by the Electro-Optics (EO) laboratory (AARI-2) bearly a block away from AFIT (bldg 622).

D.1.3 Trapix Plus 4B Image Capture. The Trapix Plus 4B Virtual Image Processing system is designed to acquire images from video sources and convert them into 10-bit per pixel, gray-scale

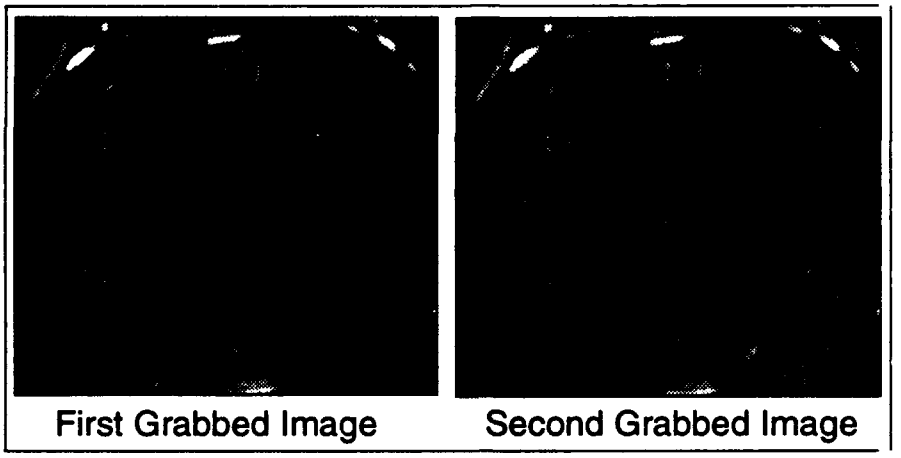

Figure D.4. Consecutive Image Grabs

values. Included as appendix A are copies of a number of brochures that describe the Trapix system. Basically, the Trapix 4B uses a 10-bit video rate digitizer to obtain medium performance digitizing. The system can be purchased in a number of configurations depending on the needs of your organization. The '4B' configuration is a four channel image processing system with 64 MBytes of RAM. The RAM is allocated to eight 8-MByte dual port image memories that can be software configured in any combination of frame sizes and bit depths. The software that supports the Trapix system is called Virtual Image Processing (VIP). VIP "provides the TRAPIX user with a set of image processing functions and utilities in the C language" according to the Recognition Concepts, Inc (RCI) brochure. What was most important to me was the system's capture rate, 32 frames per second.

Fortunately we did not have to learn more than this about the system. AARI-2 maintains a service contract for system support. One individual in particular was extremely helpful to me. Ernie Armstrong operated the system, transferred my data to Sun 8-mm tape, and discussed program design for retrieving the data and translating it into a format of use to me.

My first attempt at capturing video was instructive but not productive. My first video was simply a Super VHS recording of cars moving along National Road with the Parcoure in the foreground. Mr. Armstrong captured ten seconds of images from the tape and stored it as 8-bit per pixel binary data. Display of this data showed a cross hatch pattern covering the image. This pattern changed with time and is the result of the high frequency chromiance (color) information contained on the video tape. The second attempt to capture video was far more successful. In this attempt, we used a video made from forward looking infrared radar (FLIR) of a tank maneuvering in a field. FLIR images are gray scale only and have no color information. Mr. Armstrong converted

the video into a 10-bit per pixel format and transferred it to a Sun 8-mm tape using the Unix command for tape archive, **tar**. When we left the EO laboratory, we had a tape with 263 image files.

Each file represents a frame of 1024 x 512 pixels. At 10-bits per pixel the file size should be 655.36 KBytes long. However, Mr. Armstrong stored each pixel as two bytes or 16 bit words. Therefore, the file sizes are 1.048576 MBytes long. The video sequence we chose to capture is roughly 8.2 seconds long or 263 frames. Because the FLIR video is only 512 x 714 pixels in size, much of the approximately 1 MByte data can be discarded. Also, the data can be converted from a binary file using two bytes per pixel to 10 bits or even 8 bits to further compact the files. However, since we are not limited by disk space, we do not see a need to compress them.

Transfer of the data to 8-mm tape was done using the tape archive, **tar**, command in Unix. We intend to use these images on the Silicon Graphics IRIS 4D/440VGX computers however, the Silicon Graphics machine have no tape drive connected to them. In order to transfer my images to the Silicon Graphics workstations, we had to load the 8-mm tape onto **scgraph's** 8-mm drive and then file transfer, **ftp**, them to the Silicon Graphics workstations. We could have used the NeXt workstations but we needed more than 270 MBytes of hard disk space and the NeXt cluster did not have the room. Specifics on how to use **tar** and **ftp** can be obtained using the manual pages or **man** command in Unix.

With the data safely stored on the Silicon Graphics computers, we needed to know exactly how to access the data. That is, how is the data stored and what code would be required to read it. As stated earlier, the data is stored as the low order ten bits of a 16-bit integer value. Pixels are sequentially stored in order from the lower lefthand corner of the image to the upper righthand corner. Row one precedes row two and so on. We designed the following program to convert the data from ten bit pixels in binary format to eight bit pixels in ASCII format. The program follows.

```
#include <stdlib.h>
#include <stdio.h>
// The following line is used to identify whether the program
// is to run on the NeXt machines or the Silicon Graphics computers.
// This definition is required because the two have different integer
// sizes as defaults.  The NeXt uses a two byte integer while the
// Silicon Graphics uses a four byte integer.  However, my data is
// in binary and will not correctly be loaded if this is not set
// Right.  If you use this on a Silicon Graphics computer, change
// "NeXt" to "SGI".
#define NeXt
#ifdef SGI
# define INTSIZE short int
#else
# define INTSIZE int
#endif
```

```
main(argc, argv)
int argc;
char *argv[];
{
FILE *infile, *outfile;
          // When the data is read in, it can be place directly into
          // place holders of the correct length.  Here I use a structure
          // to parse the data in to discardable and retainable data. When
          // the structure is filled using two bytes, the high order bits
          // will fill the 'sixbit' bitfield and the low order bits fit
          // into the 'eightbit' bitfield.  Finally we get to my twobits
          // which isn't worth keeping so it too is discarded.
        struct bitfield {
             unsigned INTSIZE sixbits   : 6;
             unsigned INTSIZE eightbits : 8;
             unsigned INTSIZE twobits   : 2;
        };
        // Here I define a buffer to hold one row of the image in
        // an array of bitfields 1024 elements long.
struct bitfield pixel[1024];
        // Throw in a few counters for fun.
int i,j;
if(argc != 3) {
printf("\n Command error\n\n\t>bin2asc infile outfile\n\n");
return 0;
}
          // First I open the binary file as a read only file.  If this goes
  // smoothly, no errors, continue by opening the output ascii file.
        if ((infile = fopen(argv[1], "r")) != NULL) {
             outfile = fopen(argv[2], "w");
// Next I gather each row of 2048 elements.
             for (i=0;i<512;i++){
                fread(pixel,sizeof(pixel[0]),1024,infile);
// Since the images don't fill the entire area, I clip
// the lower 67 lines.
if (i>66) {
for (j=0;j<700;j++)
  fprintf(outfile,"%d ",pixel[j].eightbits);
        fprintf(outfile,"\n");
           }
     }
     // Time to close the files
             fclose(infile);
             fclose(outfile);
          }
          else {
           return 0;
          }
}
```

One the data is transformed into

D.2 Images as Figures

Throughout this appendix we displayed images from various sources. In most cases the images were scanned directly into the Encapsulated Postscript (EPS) format. Once in the EPS format, the images are used as figures directly in LaTeX documents. In some cases the scanned images were combined or altered using *Topdraw*. This section describes how images are scanned into the NeXT workstation and included as figures. A section is included describing the use of Topdraw to alter the images before inclusion as well as a utility that allows the NeXT user to capture screen images (GRAB).

D.2.1 Scanned Images. The scanner application is called *Scan-X.app* and is located in the **LocalApps** folder. Figure D.5 shows the user interface for this application. The steps to scanning

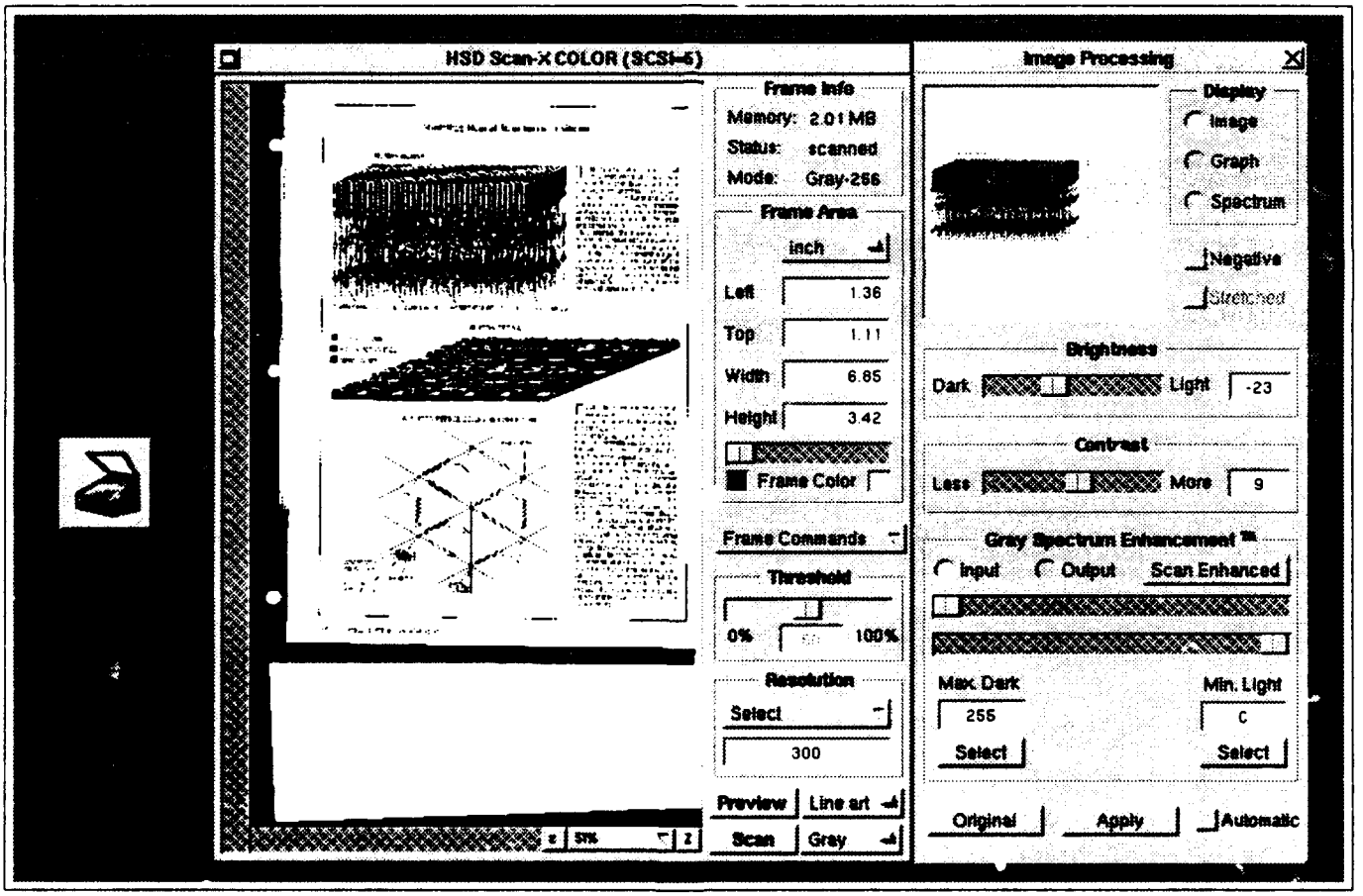

Figure D.5. Scanner User Interface [Obtained from the NeXT workstation Scan-X application using the GRAB facility]

an image into the NeXT machine are listed below.

1. Log into Bilbo and select the *Scan-X* application.
2. Place your image on the scanner, face down with the top left corner at location 0,0. The grid template surrounding the glass plate indicates 0,0 to be closest to the "HSD" logo.
3. Click on the **PREVIEW** button in the *HSD Scan-X Color* panel. The scanner will make some mechanical noises and eventually a dim image will appear in the left side of the panel.
4. The mouse can now be used to identify which portion of the preview image you want scanned. To do this, you must draw a box around a region of the image of interest in a click and drag fashion.
5. After defining the scan region, the **SCAN** button will be highlighted. A box to the right of this button allows you to identify how the image will be interpreted. The options are;
   - **Line Art**: This type of scan interprets the image as a collection of black and white pixels. Newspaper photographs and pictures in books are of this type as well as many graphs.
   - **Dither**: No, this doesn't mean stand around and act stupid, although that is about as useful as this option is on the NeXT computer. Dithering turns a half-tone picture into line art (a series of dots spaced to give the illusion of gray scale). The NeXT does this automatically.
   - **Gray**: Implying gray scale, this options assumes the picture has gray levels spanning 0 to 255 values.
   - **Color**: The scanner will reproduce color pictures but since the NeXT at AFIT can't print in color, we will not go into this option.

   When you select an option, notice that the region labeled *Frame Info* shows how much memory the image will occupy when scanned. Clicking between line art and gray scale shows a dramatic difference. Always use line art over gray if you want to keep the system administrator happy. For example, one of my images gobbled 3.5 Mbytes in gray while only taking 350 Kbytes in line art.

   A second adjustment to make is the "Resolution" option. On the NeXT machine and laser printers you cannot tell the difference between 400 dots per inch (DPI), 300 DPI, and 200 DPI. In the interest of disk conservation, choose 200 or less.
6. Click on **SCAN** and the scanner will duplicate your image in memory. When its finished, if your using gray, the image is placed in the right *Image Processing* panel.

7. If you use gray, adjust the *Brightness* and *Contrast* of the image before saving it. We suggest you play with this feature to get an understanding of its capabilities. Once you've finished adjusting the image, click on the **Apply** button to have the changes made to the image on the left.

8. Using the **Frames Commands** option box, select **size to fit**. If you are satisfied with the image, click on the **Documents** button in the main menu. Two options are available; Tagged Information File Format (TIFF) or EPS. EPS is required for direct introduction into LaTeX files. Either will work in the Topdraw application discussed later. After supplying a name, the file is placed in your directory.

D.2.2 Grabbing Images. The image capture program is called GRAB and is located in the **Demos** folder. GRAB is a simple program to use. The menus of importance are shown in Figure D.6.Once you have the image you want displayed on the screen, start GRAB. Under the

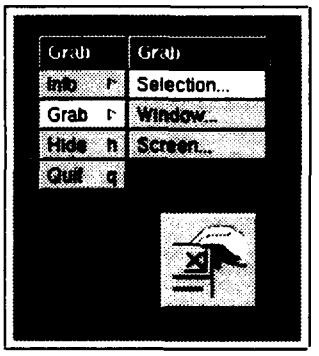

Figure D.6. Grab Menus

Grab button of the menu is a list of three options. We have only mastered one. That is the **Selection** button. This button replaces the cursor with an angle bracket. Click on the point of the screen to the upper left of the figure you want. Drag the cursor to the lower right hand corner of the figure and release. Grab will churn for a while and eventually will present the save panel and you can assign the image a name. Grab will only save in the TIFF format. To convert to the EPS format, double click on the file. This brings it into the viewer where you can save it to an EPS file format.

D.2.3 Altering Images. Once you've captured the images you need, they can be altered using Topdraw. Topdraw is a full featured drawing package but we do not intend to address all its capabilities. We are only interested in it for the specific use of importing an image and adding additional features. Topdraw is located in the **LocalApps** folder.

1. Start the Topdraw application.
2. Select **Window** and **New Document**. This opens a blank document where you will place the imported image. With a blank document open, the **Import** and **Export** options become available for selection.
3. Under the same window submenu, select **Import** to bring the image into your document. When you do so, a panel that requests how the image will be brought into the document is presented. Select the option for copying it directly in and not the reference option.
4. Topdraw places a box around the image. If you don't want this box change the line color to white before importing the image. The box will still be there but its invisible. We recommend you do this because LaTeX does not handle the boxes very well.
5. Make your alterations. Play with the program and get to know its capabilities. Topdraw is much like other drawing packages found on other computers.
6. When you are ready to save the document, select **Export**. A save panel gives you the option to save the image in three file formats;*EPS*, *TIFF*, and the Topdraw Format. Select **Selected Objects** and **EPS** and supply a filename.

D.2.4 Importing Figures into LaTeX. The figures you've saved either by Scanning, Grabbing, or Drawing can now be imported into LaTeX as long as they are in EPS format. First you need to tell the LaTeX compiler that your using EPS files with the inclusion of the style **epsf** in your document line.

```
\documentstyle[12pt,epsf,oper511]{article}
\newcommand {\epsbox}[1]{\centerline{\framebox{\epsffile{#1}}}}
```

Also include a *newcommand* definition to place a box around each figure and center the figure. Then when you want to place a figure in your document the following lines are required (the whole block for each figure).

```
\begin{figure}[htbp]
\epsfysize=5.00in
\epsfxsize=3.00in
\epsbox{eps/scanned.eps}
\caption[Scanner User Interface]{Scanner User Interface [Obtained
from the NeXT workstation Scan-X application using the GRAB
facility]}\label{FIG:Scanner}
\end{figure}
```

The first line heralds the beginning of the block with the location options in square brackets to the right. The symbols h, t, b , and p tell LaTeX to place the figure *h*ere, *t*op of the next page,

*b*ottom of the next page, or on a *p*age of figures. It is important to understand that if all three of these options are not specified, LaTeX will put this and all subsequent figures in the back of the paper. The next two lines identify a region size in the *Y* and *X* directions to put your figure. You do not need both *epsfysize* and *epsfxsize*. If you only use the *X* size specifier, LaTeX will size the figure to fit the vertical dimension of the figure in the space identified in *in* (inches), *cm* (centimeters), or *pt* (points). One thing to note here, the EPS file may have a description of the image that is rotated before being displayed. In this case, LaTeX can screw-up and size the image incorrectly. If this happens, change the epsfxsize so the horizontal dimension is proportionate to the room you wish to fill vertically. The macro line containing *epsbox* identifies which file to include for the figure space and to place a box around the figure. In this example, the file *scanned.eps* is located in the **EPS** folder (directory). The *caption* line gives a Table of Contents caption in brackets and the actual figure caption in braces. This is followed by the *label* command. Labels are used to reference figures, tables, and equations to textual material through the *ref* (reference) command. An example of the reference command is given below.

```
Figure~\ref{FIG:Scanner}
```

Before leaving this topic we want to mention two important problems that with drive you nuts until you solve them. First, when you run LaTeX on your document file, you should have your images available in the location indicated by the epsfile command. If you don't (for any number of reasons), LaTeX will not properly center and scale the figures giving unpredictable results (i.e. overlay the caption with the figure).

Second, images scanned-in and saved directly as *EPS* files appear offset in the document. This is particularly evident when a box is placed around the figure. This problem can be solved by importing the figure into Topdraw and then exporting back to an *EPS* file. We believe this changes the header information which is incorrectly set by the scan software. Below are the pertinent lines of the header set by the scan software. Notice the negative values in the third and fourth lines. Files exported by Topdraw do not have values less then zero.

```
%%For: dswanson
%%DocumentFonts: (atend)
%%Pages: 0 -1
%%BourdingBox: 3 -4 338 230
%%EndComments
```

D.3 Important Points of Contact.

This section is so that others who follow can annoy various people throughout the United States. It is a list of who we talked to and how we contacted them.

- *Dave Andes* - works for the Navy at China Lake Naval Weapons Laboratory. He is associated with the MINND program which uses a Silicon Graphics machine connected to several pairs of ETANN boards.

  Phone number: (AV) 437-3848 E-Mail: andes@scfb.nwc.navy.mil

- *Misha Mahowald* - A PhD student at the California Institute of Technology (Caltech). Is an assistant to Dr. Carver Mead and played an important role in developing the Silicon Retina.

  Phone: (818) 356-6871 E-Mail: misha@hobiecat.Caltech.edu

- *Tom Rathbun* - A point of contact with SDIO. Tom is a recent graduate of AFIT and is sponsoring my work with motion detection. He is responsible for a contract with Amber Engineering. Amber Engineering is developing the Neuromorphic Infrared Focal Plane Chip that is based on Mead's Silicon Retina. Tom's organization is the Guided Interceptor Branch (WL/MNSI) and is looking for missile seekers.

  E-mail: RATHBUN@eglin.af.mil

- *Allen Waxman* - A scholar and a gentleman, Dr. Waxman has published many papers concerning motion segmentation.

  E-mail waxman@xn.ll.mit.edu

- *Brian Yasuda* - works in the Electro-Optics laboratory in building 622 (across from AFIT park). His organization (AARI-2) has a Trapix 4B system (made by RCI) used to capture images from video tape.

  Phone: 5-9615

Bibliography

Amrine, John M. et al. *Project Ares: A Systems Engineering and Operations Architecture for the Exploration of Mars.* Report for the Department of Operational Sciences, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH. March 1992 (Report submitted to DTIC and is in the AFIT library collection.)

Booch, G. *Object Oriented Design with Applications.* New York: The Benjamin/Cummings Publishing Company Inc., 1991.

Descour, M. *Functional Aspects of the Retina Relating to Infrared Focal Plane Arrays.* Report by the Optical Sciences Center, Optical Detection Laboratory, University of Arizona, Tucson, AZ, April 1992.

Goldman-Rakic, Patricia S. "Working Memory and the Mind" *Scientific American, 267:* 111-117 (September 1992)

Grzywacz, Norberto M. and Tomaso Poggio, *An Introduction to Neural and Electronic Networks.* Edited by Steven F. Zornetzer et al. San Diego: Academic Press, Inc. 1990.

Grossberg, S. "Nonlinear Neural Networks: Principles, Mechanisms, and Architectures," *Neural Networks (in press)* 1-61 (1987).

Kreyszig, E. *Advanced Engineering Mathematics.* New York: John Wiley and Sons, 1988.

Levine, Martin D. *Vision in Man and Machine.* New York: McGraw-Hill Book Company, 1985.

Mahowald, Misha A. and Carver A. Mead, "The Silicon Retina," *Scientific American, 264:* 76-82 (May 1991).

Masland, Richard H. "The Functional Architecture of the Retina," *Scientific American, 255:* 102-111 (December 1986).

Massie, M. A. et al. "Neuromorphic Infrared Focal Plane Performs On-plane Local Contrast Enhancement, Spatial, and Temporal Filtering." Report to Strategic Defense Initiative Personnel. WL/MNSI, Eglin AFB, January 1992.

Mead, Carver A. and M. A. Mahowald. "A Silicon Model of Early Visual Processing," *Neural Networks 1:* 91-97 (1988).

Öğmen, H. and S. Gagné. "Neural Network Architectures for Motion Perception and Elementary Motion Detection in the Fly Visual System," *Neural Networks,3:* 487-505 (May 1990).

Oppenheim, Alan V. *Discrete-Time Signal Processing.* New Jersey: Prentice Hall, 1989.

Poggio, T. and Christof Koch. "Synapses That Compute Motion," *Scientific American, 256:* 46-52 (May 1987).

Press, William H. *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1988.

Ratliff, F. "Inhibitory Interaction and the Detection and Enhancement of Contours." In Rosenblatt, editor, *Sensory Communication* MIT press, 1959.

Reid, J. G. *Linear System Fundementals*, New York: McGraw-Hill, Inc, 1983.

Snyder, Solomon H. "The Molecular Basis of Communication between Cells," *Scientific American, 254:* 20-33 (October 1985).

Stevens, Charles F. "The Neuron," *Scientific American, 241:* 54-73 (March 1979).

Swanson, David E. *Using Object Oriented Design to Simulate a Massively Parallel Architecture of Neurons*, Masters Thesis, School of Engineering, University of Colorado, Colorado Springs, Colo. 80910 (December 1992).

Synthesis Group on America's Space Exploration Initiative. *America at the Threshold,* Washington: Government Printing Office, 1991.

Werblin, Frank S. "The Control of Sensitivity in the Retina," *Scientific American, 228:* 70-79 (January 1973).

Wilsey, David G. "U.S. National Launch Systems Effect on Assured Access to Space," Report for the Department of Operational Sciences, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH. March 1992.

Young, David M. and Robert Todd Gregory. *A Survey of Numerical Mathematics*, Addison-Wesley Publishing Company, Reading Mass. 1972.

Vita

Captain David Swanson is a degree candidate for a masters of Space Operations at the Air Force Institute of Technology (AFIT). He is also a candidate for a masters of Electrical Engineering at the University of Colorado in Colorado Springs.

In 1986 Captain Swanson received his B.S.E.E. from Southern Illinois University at Carbondale Illinois where he was honored as the Outstanding Senior Engineer. During his collegiate years he held the offices of chairmen and vice-chairmen of the student chapter of Institute of Electrical and Electronics Engineers (IEEE) as well as chairmen for the Engineering School's Joint Student Council. He was commissioned through the Air Force Reserve Officer Training Corp (AFROTC) at Southern Illinois University in 1986 and attended Undergraduate Space Training enroute to his first assignment with the Consolidated Space Testing Center operating location AB. Captain Swanson spent his first year of active duty in Lockheed Technical Operations Training, training to be a Satellite Planner Analyst on the Fleet Satellite Communications constellation.

Captain Swanson was instrumental in activating the 3d Satellite Control Squadron (3SCS), a unit in the 2d Space Wing (2SWG) located at Falcon AFB, Colorado. From 1988 to 1991, Captain Swanson was certified as both a Planner Analyst and a Mission Controller in the 3SCS. Before coming to AFIT, Captain Swanson worked in 2SWG/DOT where he defined requirements for and installed simulators used to train mission control crews.

Captain Swanson was married in 1987 to the former Miss Cynthia L. Alshouse. Their union flourishes in the form of two children - Douglas Edward and Lauren Michelle.

Permanent address: 740 Liberty Bell Ln.
Libertyville, Il.
60048

December 1992 Master's Thesis

Retinal Modeling: Segmenting Motion from Spatio-Temporal Inputs using Neural Networks.

David E. Swanson

Air Force Institute of Technology, WPAFB OH 45433-6583

AFIT/GSO/ENG/92D-04

Captain Thomas Rathbun
SDIO - Guided Interceptor Branch
WL/MNSI
Eglin AFB, FL

We applied two first-order, linear, time-varying, differential equations to the task of segmenting motion from sequences of images. The equations are modified Grossberg formulas for long-term and short-term memory models characterizing the neurotransmitter and cell-activity levels of a synapse and neuron. We described how a two layered, sensory, neural network can be built using the equations to simulate the amacrine neurons of the retina. The model is defined using adaptive input nodes (adaptive model) and is compared to a similar model without these nodes (O&G model). By replicating the basic amacrine neuron model to form both one- and two-dimensional arrays, we created a novel method for processing images over time and space. To simulate the veto effect observed in shunt inhibitory synaptic junctions, we applied a nonrecurrent, asynchronous, inhibitory region in the receptive field of our amacrine neural model. We show how this effects the performance of the model in one dimension. In two dimensions we investigate the models' response to synthesized imagery (pristine) and to real, forward looking infrared radar (FLIR) images. The output of our models are further processed through two types of moving-average filters – causal and noncausal.

Motion Segmentation, Image Processing, Sensors, Artificial Neural Networks, Retinal Modeling, Moving Average Filter, Vision

187

UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL